ISBN 978-83-66678-24-8
ISSN 1897-2691

Dr hab. inż. Paweł Kossakowski jest profesorem Politechniki Świętokrzyskiej w Kielcach. Pracuje w Katedrze Wytrzymałości Materiałów i Analiz Konstrukcji Budowlanych na Wydziale Budownictwa i Architektury. Główna tematyka jego działalności naukowej dotyczy materiałów konstrukcyjnych stosowanych w budownictwie (drewno i stal) oraz nowoczesnych technologii komputerowego wspomagania projektowania inżynierskiego. Jest autorem i współautorem ponad 160 prac naukowych obejmujących artykuły publikowane w wielu renomowanych zagranicznych i krajowych czasopismach naukowych i naukowo-technicznych, referaty konferencyjne, jak również rozdziały w monografiach. Dr hab. inż. Paweł Kossakowski jest także autorem i współautorem kilku skryptów dydaktycznych.

Autor jest członkiem krajowych i zagranicznych towarzystw naukowych i technicznych, rad naukowych czasopism oraz organizacji branżowych.

W trakcie swojej ponad dwudziestoletniej praktyki inżynierskiej związanej głównie z projektowaniem konstrukcyjno-budowlanym dr hab. inż. Paweł Kossakowski opracował oraz zweryfikował szereg projektów w tym zakresie, jak również sporządził wiele opinii i ekspertyz o stanie technicznym różnych obiektów budowlanych. Jest rzeczoznawcą budowlanym w specjalności konstrukcyjno-budowlanej. Łączna ilość opracowań technicznych, których jest autorem bądź brał udział w ich opracowaniu, wynosi ponad 600.

Tematyka książki jest bardzo aktualna, gdyż dotyczy w ogólności poszukiwania jak najlepszych technologii i narzędzi wspomagających proces projektowania konstrukcji budowlanych. Taką technologię zdaniem Autora stanowi projektowanie oparte na programowaniu wizualnym, niewymagającym od projektanta branży budowlanej dodatkowej wiedzy programistycznej.

W opracowaniu opisano szczegółowo środowisko pracy systemu Dynamo, interfejs, najczęściej używane węzły, obiekty skryptu i operatory, jak również elementarne obiekty graficzne. Na uwagę zasługuje fakt przedstawienia dziesięciu praktycznych zastosowań opracowanych skryptów wizualnych do kształtowania formy i konstrukcji obiektów. Podchodząc do zagadnienia w sposób szeroki i kompleksowy, pokazano możliwość integracji środowiska Dynamo z innymi systemami służącymi do modelowania i obliczania, jak również optymalizacji obiektów budowlanych.

Tematyka monografii jest aktualna, gdyż pokazuje nowe możliwości, jakie daje zastosowanie nowoczesnych narzędzi cyfrowych w projektowaniu, a poziom merytoryczny dzieła jest wysoki. Recenzowana praca, według wiedzy recenzenta, jest pierwszym krajowym opracowaniem ujmującym zastosowania programu wizualnego Dynamo do kształtowania konstrukcji budowlanych w sposób całościowy i kompletny. Na rynku krajowym brak jest tego rodzaju opracowania.

Książka może być nie tylko pomocą dydaktyczną dla studentów kierunku budownictwo, ale może być również wykorzystana przez inżynierów budownictwa w ich działalności projektowej.

Fragment recenzji
dr hab. inż. Jolanty Dźwierzyńskiej, prof. PRz

Recenzowana monografia dotyczy projektowania opartego na programowaniu wizualnym. Jak sam autor zauważa we wstępie, w kraju nie ma podobnej publikacji całościowo ujmującej projektowanie wizualne. Monografia z pewnością przyda się studentom wydziałów budownictwa oraz inżynierom pragnącym poszerzyć wiedzę z tej dziedziny.

Za najważniejsze oryginalne osiągnięcia naukowe Autora, stanowiące wkład w dyscyplinę inżynieria lądowa i transport, uznaję:

- *zebranie w jednej publikacji wielu cennych informacji z dziedziny projektowania wizualnego,*
- *opis środowisk opartych na językach VPL,*
- *szczegółowe przedstawienie możliwości programu Dymamo wraz z podaniem przykładów zastosowań.*

Recenzowana monografia jest pozycją bardzo wartościową. Na rynku wydawniczym nie ma obecnie podobnej publikacji.

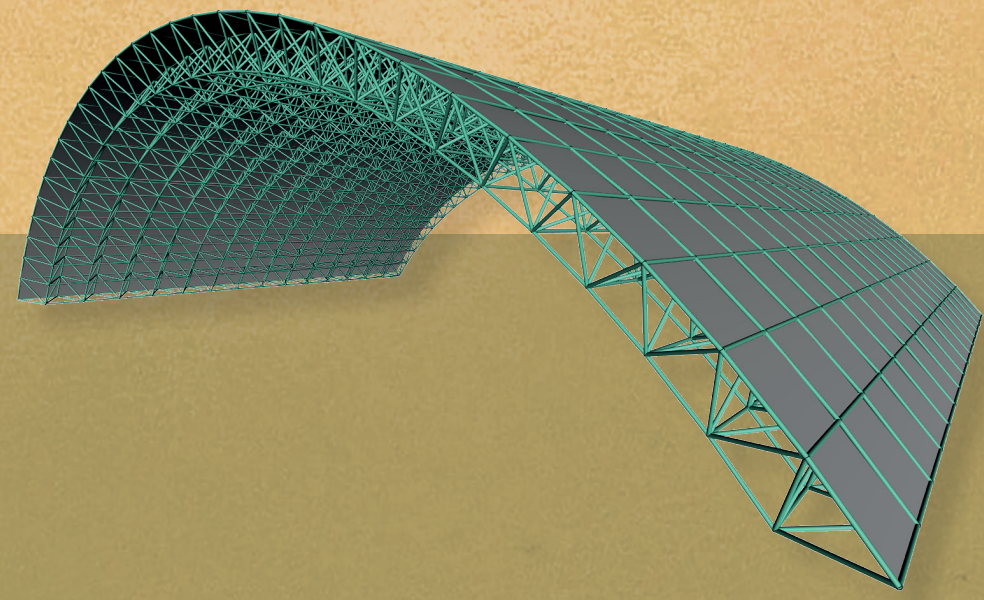
Fragment recenzji
prof. dr. hab. inż. Łukasza Drobca

Paweł Kossakowski ZASTOSOWANIE PROGRAMOWANIA WIZUALNEGO W PROJEKTOWANIU KONSTRUKCJI BUDOWLANYCH

MONOGRAFIE
STUDIA
ROZPRAWY
M148

Paweł Kossakowski

ZASTOSOWANIE PROGRAMOWANIA WIZUALNEGO W PROJEKTOWANIU KONSTRUKCJI BUDOWLANYCH



Politechnika Świętokrzyska
Kielce 2022

MONOGRAFIE
STUDIA
ROZPRAWY
M148

Paweł Kossakowski

ZASTOSOWANIE PROGRAMOWANIA WIZUALNEGO W PROJEKTOWANIU KONSTRUKCJI BUDOWLANYCH

Kielce 2022

MONOGRAFIE, STUDIA, ROZPRAWY NR M148
Redaktor Naukowy serii
BUDOWNICTWO
prof. dr hab. inż. Jerzy WAWRZEŃCZYK

Recenzenci
prof. dr hab. inż. Łukasz DROBIEC
dr hab. inż. Jolanta DŹWIERZYŃSKA, prof. PRz

Redakcja
Aneta STARZYK

Projekt okładki
Tadeusz UBERMAN

Wydanie monografii finansowane w ramach projektu z programu Ministra Edukacji i Nauki pod nazwą „Regionalna Inicjatywa Doskonałości" w latach 2019-2022, nr projektu 025/RID/2018/19, kwota finansowania 12 000 000 zł

Ministerstwo
Edukacji i Nauki

ISBN 978-83-66678-24-8
ISSN 1897-2691

Wydawnictwo Politechniki Świętokrzyskiej
25-314 Kielce, al. Tysiąclecia Państwa Polskiego 7
tel. 41 34 24 581
e-mail: wydawca@tu.kielce.pl
www.wydawnictwo.tu.kielce.pl

Spis treści

Od Autora

Dynamiczny rozwój techniki, jaki nieustająco zachodzi od końca XVIII wieku, zaowocował powstaniem i wdrożeniem wielu rozmaitych i coraz bardziej wyrafinowanych urządzeń i technologii. W szeroko traktowanej inżynierii fundamentalne było opracowanie, zbudowanie, a przede wszystkim wprowadzenie do powszechnego użytku komputerów.

Budownictwo dość wcześnie postawiło na komputeryzację, która przez długi czas obejmowała przede wszystkim etap projektowy, by obecnie objąć cały proces realizacji inwestycji budowlanej. Poza standardowymi i ugruntowanymi już systemami wspomagającymi projektowanie interesującym narzędziem w tym zakresie jest wykorzystywanie środowisk programistycznych. Niestety są one wciąż stosowane przez projektantów w marginalnym stopniu, głównie z uwagi na barierę, jaką dla inżyniera budowlanego stanowi samo programowanie. Naprzeciw temu wychodzi programowanie oparte na kodzie wizualnym, który od inżyniera projektanta nie wymaga specjalistycznej wiedzy, jak ma to miejsce podczas tworzenia tradycyjnych skryptów tekstualnych. W dodatku środowiska programowania wizualnego stosowane w branży architektoniczno-budowlanej są wyposażone w szereg opcji i narzędzi wydatnie automatyzujących, upraszczających i przyspieszających proces projektowania, a nawet zarządzania inwestycją.

Właśnie technologii opartej na programowaniu wizualnym jako nowoczesnemu narzędziu wspomagającemu projektowanie konstrukcji budowlanych poświęcona jest niniejsza publikacja. Na tle krótkiego tła historycznego opisującego komputeryzację procesu produkcji zaprezentowano w niej systemy wspomagania projektowania oraz koncepcję i narzędzia programowania wizualnego. Przybliżono środowiska programistyczne używane w branży architektoniczno-budowlanej, szczegółowo omawiając program Dynamo, który jest jednym z podstawowych i często stosowanych programów w projektowaniu konstrukcyjnym. Aby jak najlepiej zaprezentować możliwości tego środowiska, przygotowano i szczegółowo omówiono przykłady praktycznego zastosowania programowania wizualnego w kształtowaniu zarówno prostych, jak i bardziej skomplikowanych struktur konstrukcyjnych oraz integracji z innymi systemami modelowania i obliczania, a także optymalizacji obiektów budowlanych. Omówiono również przeglądowo możliwości wykorzystania programowania wizualnego oraz kierunki jego rozwoju w branży architektoniczno-budowlanej.

Głównym przyczynkiem podjęcia tematu w takim ujęciu był brak publikacji, które traktowałyby zagadnienie w sposób szeroki i kompletny, szczególnie w ujęciu zastosowania programowania wizualnego w projektowaniu konstrukcji budowlanych. Jak do tej pory technologia ta jest dość dobrze i szeroko opisana w zakresie szeroko rozumianego designu oraz w pozycjach poświęconych architekturze. Tym samym niniejsza monografia jest pierwszą, która kompleksowo podchodzi do kwestii zastosowania środowiska programowania wizualnego w branży konstrukcyjnej.

Inną kwestią jest brak profesjonalnie przygotowanych publikacji dydaktycznych, które są niezbędne w procesie edukacyjnym.

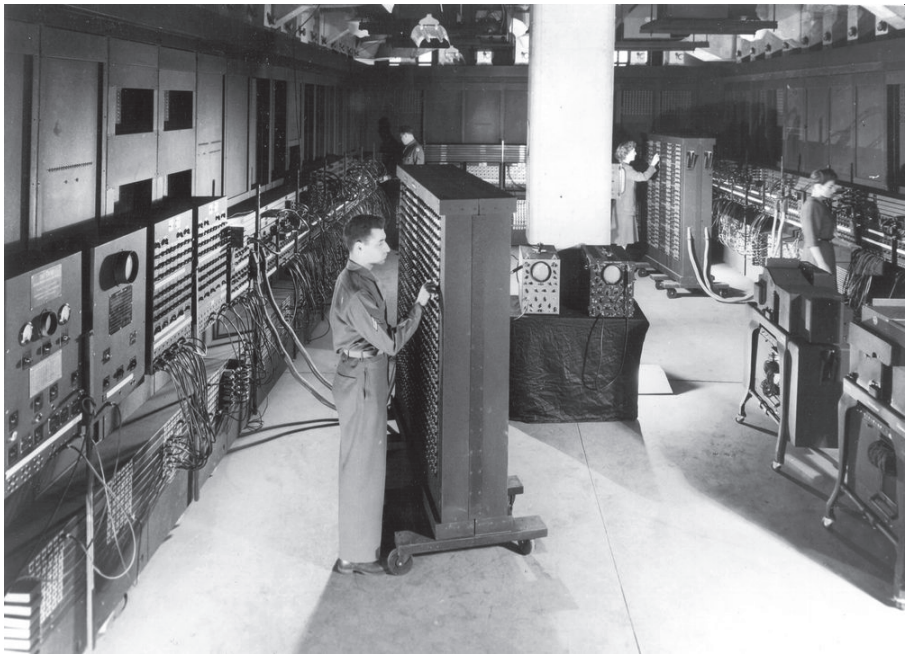
Życząc miłej lektury, Autor żywi głęboką nadzieję, że Szanowni Czytelnicy, którzy nie mieli do tej pory do czynienia z programowaniem wizualnym, zainteresują się nim szerzej i wykorzystają dostępne narzędzia w swojej praktyce inżynierskiej.

Kielce, 7 kwietnia 2022 *Paweł Kossakowski*

1 WPROWADZENIE

1.1. KOMPUTERYZACJA PROCESU PRODUKCJI

Choć niektórym może się wydawać, że komputery to relatywnie młode urządzenia, to ich historia sięga aż 80 lat. Wciąż żywe są dyskusje, którą maszynę można uznać za pierwszy na świecie komputer – czy, jak się to najczęściej przyjmuje, był to ENIAC skonstruowany w USA w latach 1943-45 [1, 2], czy brytyjski Colossus z roku 1943 [3], a może jednak ABC skonstruowane w roku 1939 przez Johna Atanasoffa i Clifforda Berry'ego [4] lub Z3 niemieckiego pioniera Konrada Zuse z roku 1941 [3]? Podstawowym problemem w tym zakresie są budowa i funkcjonalność danego urządzenia, tak aby spełniało ono kryteria uznania go za komputer, a nie zaawansowany kalkulator. Nie zmienia to faktu, że właśnie lata 40. XX wieku to okres, kiedy urządzenia nabierające cech komputerów, a więc programowalnych, wielozadaniowych maszyn obliczeniowych zdolnych do przetwarzania informacji, zaczęły powstawać i stopniowo znajdować coraz szersze zastosowanie [5]. Pierwsze komputery były ogromnych, wręcz gigantycznych rozmiarów (rys. 1.1), o czym może choćby świadczyć nazwa jednego z nich ochrzczonego jako Colossus.



Rys. 1.1. Komputer ENIAC [6]

W latach 40. i 50. XX wieku trwały intensywne prace w zakresie budowy komputerów oraz systemów służących do ich programowania [7]. Skonstruowano kolejne maszyny, optymalizując opracowane do tej pory rozwiązania.

W kolejnych kilkudziesięciu latach następował progres w zakresie mocy obliczeniowych konstruowanego sprzętu, który pierwotnie działał w oparciu o lampy elektronowe, by w kolejnych latach bazować na elementach i układach półprzewodnikowych. Wydatnie przyczyniło się to do redukcji gabarytów, a także zapotrzebowania na energię kolejnych maszyn. Prawdziwym przełomem był jednak okres lat 70. i 80. XX wieku, kiedy to wprowadzono pierwszy komputer osobisty, czyli tzw. PC (PeCet) od ang. *Personal Computer*. Powszechnie uznaje się, że był to Altair 8800 [8] opracowany pod koniec roku 1974 i wprowadzony na rynek na początku 1975 roku. Został on ochrzczony właśnie terminem *Personal Computer* przez swojego twórcę, Eda Robertsa, jednakże wielu za pierwszego PeCeta uznaje maszynę KENBAK-1 opracowaną w roku 1970 i wprowadzoną na rynek w roku 1971 [9, 10]. Altair 8800 w wersji podstawowej nie miał klawiatury, monitora i pamięci masowej. Należało więc doposażyć go w wymienione urządzenia peryferyjne, co stanowiło dużą niedogodność. Dopiero wypuszczone na rynek w roku 1977 maszyny typu Commodore PET, RadioShack TRS-80 i Apple II były w pełni wyposażonymi urządzeniami, które przypominają w dużym stopniu sprzęt wykorzystywany obecnie. Od tego momentu można więc datować intensywny rozwój komputerów, które dzięki dostępności dla zwykłych użytkowników, zaczęły stawać się urządzeniami wykorzystywanymi w wielu obszarach i dziedzinach życia, niekoniecznie związanych z pracą zawodową. Obecnie komputery osobiste stanowią jeden z szerokiego segmentu tego typu urządzeń, bo coraz częściej używane są laptopy, tablety czy nawet smartfony do wykonywania całkiem skomplikowanych zadań i czynności, mocą obliczeniową i możliwościami przewyższając stosowane całkiem jeszcze niedawno jednostki klasy PC.

Obecnie trudno znaleźć dziedzinę życia, która nie uległaby postępującej od wielu już dziesięcioleci komputeryzacji. Najintensywniejszy rozwój w tym zakresie następował w systemach obliczeniowych, gdzie dzięki zastosowaniu komputerów nieporównywalnie wzrosła szybkość obliczeń, a co za tym idzie możliwości rozwiązywania wielu problemów. Osobną gałęzią są metody numeryczne, które znajdują zastosowanie utylitarne, przede wszystkim w modelowaniu komputerowym. Ich aplikacje obejmują praktycznie wszystkie dziedziny techniki, ale również i inne branże czy obszary, takie jak choćby medycyna.

Komputeryzacja w naturalny sposób wkroczyła również do szeroko traktowanego projektowania inżynierskiego. Podstawową kwestią była tutaj możliwość programowania, która pozwalała na tworzenie zaawansowanych procedur, za pomocą których przeprowadzano symulacje analizowanych procesów. Drugim niezmiernie ważnym czynnikiem była możliwość wykonywania skomplikowanych i czasochłonnych obliczeń. Ich wyniki uzyskiwane były w relatywnie krótkim czasie, co dodatkowo umożliwiało analizę wielowariantową, weryfikację różnych

scenariuszy, a tym samym optymalizację podejmowanych zagadnień. Przykładowo komputerowe obliczenia numeryczne były stosowane w wielu pionierskich symulacjach prób w ramach prowadzonych programów kosmicznych. Biorąc jednakże pod uwagę możliwości techniczne w tamtym czasie, były to aplikacje bardzo specjalistyczne i wymagające posiadania zaawansowanego i niedostępnego na otwartym rynku sprzętu. Momentem przełomowym było upowszechnienie komputerów, ale przede wszystkim skonstruowanie i upowszechnienie komputerów osobistych klasy PC, co zaczęło mieć miejsce od lat 80. XX wieku. Stworzone zostały wtedy maszyny, które odpowiednio oprogramowane, zaczęły być sukcesywnie wykorzystywane w coraz to różnych obszarach życia, a podstawowym z nich była oczywiście inżynieria. W tym miejscu należy zaznaczyć, że nawet w naszym kraju komputeryzacja i informatyzacja zostały dość wcześnie wprowadzone do systemu edukacyjnego, bo zajęcia z tzw. informatyki były włączane do programów nauczania szkół średnich już w latach 80.

Wraz z rozwojem programów pozwalających na automatyzację procesu przygotowania produkcji pojawiła się szansa na ich zastosowanie na etapie opracowywania koncepcji, prototypowania i wreszcie projektowania produktu. Efektem tego było opracowanie systemu tzw. komputerowo zintegrowanego wytwarzania, określanego akronimem CIM od angielskiego terminu *Computer Integrated Manufacturing*. System ten obejmuje komputerowe wspomaganie wszystkich etapów produkcji, tj. samego procesu wytwarzania, jak i jego logistyki. Istotnymi elementami CIM są automatyzacja i robotyzacja procesu, które w połączeniu z komputeryzacją pozwalają na elastyczne dostosowywanie i optymalizację procesu produkcji w warunkach dynamicznie zmieniających się potrzeb rynku. Ważnym czynnikiem jest tutaj integracja (zawarta w samej nazwie) poszczególnych narzędzi z bazą, a dokładniej modelami i profilami danego przedsiębiorstwa. Takie podejście wymaga oczywiście kosztownych inwestycji w sprzęt i infrastrukturę, a także w wysokospecjalistyczną kadrę inżynierską i zarządczą. Obserwując jednakże przyspieszenie, jakie następuje w modernizacji i unowocześnianiu przedsiębiorstw produkcyjnych praktycznie na całym świecie, wydaje się, że produkcja oparta na CIM to trwały trend, od którego praktycznie nie ma odwrotu.

System CIM to tak na prawdę ogólna koncepcja wspomagania komputerowego procesu produkcji, gdzie podstawowym i istotnym elementem jak samo zastosowanie komputerów i robotów jest właśnie integracja tych narzędzi z infrastrukturą przedsiębiorstwa. W ramach CIM funkcjonuje kilka tzw. podsystemów przeznaczonych do wspomagania poszczególnych zakresów i etapów produkcji. Najważniejsze z nich to:

- komputerowo wspomagane prace inżynierskie – CAE (ang. *Computer Aided Engineering*);
- komputerowo wspomagane projektowanie – CAD (ang. *Computer Aided Design*);

- komputerowo wspomagane planowanie – CAP (ang. *Computer Aided Planning*);
- komputerowo wspomagane wytwarzanie – CAM (ang. *Computer Aided Manufacturing*);
- komputerowo wspomagana kontrola jakości – CAQ (ang. *Computer Aided Quality Control*);
- planowanie i sterowanie produkcją – PPC (ang. *Production Planning and* Control);
- planowanie zasobów przedsiębiorstwa – ERP (ang. *Enterprise Resource Planning*).

Jak widać, pięć pierwszych podsystemów obejmuje procesy ściśle związane z cyklem produkcyjnym i technologicznym, natomiast dwa ostanie dotyczą zakresów planowania i zarządzania przedsiębiorstwem. Wymienione powyżej podsystemy komputerowego zintegrowanego wytwarzania (CIM) funkcjonują już od kilku dekad w inżynierii i trudno sobie wyobrazić pracę w innym formacie.

Inżynierów uczestniczących bezpośrednio w produkcji szczególnie interesują systemy CAE, CAM i CAD, gdyż dają one narzędzia pozwalające na automatyzację szeregu procesów i czynności. System CAE to nic innego jak zestaw narzędzi w postaci oprogramowania komputerowego, służący – jak sama nazwa wskazuje – do komputerowego wspomagania prac inżynierskich. Obejmują one etap obliczeń, podczas których *de facto* weryfikowane i projektowane są zarówno elementy składowe, jak i całe produkty. Najczęściej zastosowanie znajdują tu programy numeryczne bazujące na metodzie elementów skończonych (MES), ale nie tylko. Oprogramowanie używane w systemach CAE oferuje także narzędzia analityczne do symulacji różnorakich procesów, związanych choćby z obróbką materiału, formowaniem produktu czy wreszcie optymalizacją cyklu produkcyjnego. Z kolei system CAM to platforma integracyjna, gdzie wspomagane jest połączenie i uzupełnienie fazy projektowania i wytwarzania produktu. Istotnym zakresem funkcjonowania systemu i programów CAM jest obróbka numeryczna realizowana za pomocą obrabiarek (skrót NC z ang. *Numerical Control*), gdzie oprogramowanie CAM jest podstawowym narzędziem.

1.2. SYSTEM CAD

Najlapidarniej definiując ideę CAD, należy stwierdzić, że opiera się ona na koncepcji cyfrowego odwzorowania tworzonego projektu. Podstawą jest tutaj cyfrowe modelowanie geometryczne, oparte na podstawowych obiektach takich jak punkt, linia, krzywa itp. Każdy z elementów odwzorowujących projektowany produkt składa się z wyżej wymienionych składowych obiektów geometrycznych i jest definiowany w sposób graficzny, specyficzny dla danego programu. Projektowany element konstrukcyjny zawiera w sobie tak naprawdę trzy podstawowe informacje dotyczące jego cech geometrycznych, dynamicznych oraz materiałowych czy szerzej technologicznych. Możliwa jest definicja elementów dwu- i trójwymiarowych (2D, 3D).

Elementy te mogą stanowić cały projekt, jak również jego poszczególne części, sekcje czy widoki. System CAD znalazł zastosowanie w inżynierii mechanicznej, elektrycznej, architektonicznej, budowlanej czy instalacyjnej. Obecnie trudno wymienić obszar projektowania inżynierskiego, w którym nie ma wspomagania CAD.

W uzupełnieniu należy również dodać, że z uwagi na zazębianie się zakresu projektowania i obróbki materiału i półproduktów, jakie ma miejsce nie tylko w inżynierii mechanicznej, zaczęto produkować oprogramowanie obsługujące właśnie te fazy procesu produkcji. Są one określane jako mieszane systemy CAD/CAM.

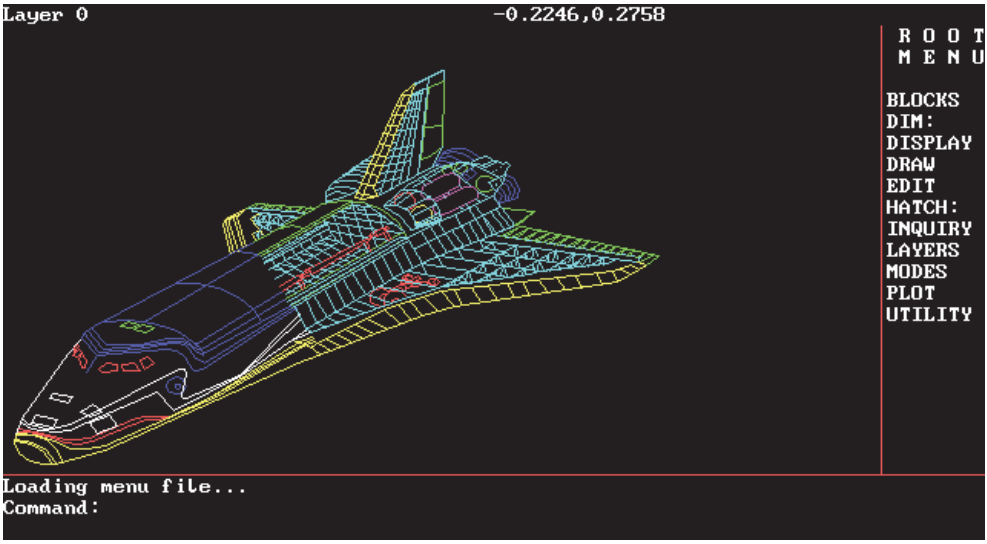
Historia systemu CAD sięga roku 1957, kiedy to dr Patrick J. Hanratty opracował system komputerowy o nazwie PRONTO [11], który umożliwiał programowanie i sterowanie numeryczne. Z tego powodu nazywany jest on niekiedy ojcem CAD/CAM. Był to krok milowy, a raczej punkt zwrotny, który umożliwił zapoczątkowanie procesu rozwoju systemów i środowisk komputerowego wspomagania procesów produkcji.

W styczniu 1963 roku doktorant Ivan Sutherland przedstawił swoją pracę zatytułowaną *Sketchpad: A Man-Machine Graphical Communication System*, w której opisał opracowany przez siebie interaktywny komputerowy system wspomagania projektowania o nazwie Sketchpad [12, 13]. Jest on uważany za pierwszy interaktywny program typu CAD, który oferował użytkownikowi możliwość wprowadzania danych za pomocą interfejsu graficznego, w tym przypadku przy użyciu pióra świetlnego na monitorze. Na rysunku 1.2 widać Ivana Sutherlanda podczas pracy w Sketchpadzie na komputerze TX-2 w MIT Lincoln Labs.

Rys. 1.2. Ivan Sutherland podczas pracy w programie Sketchpad [14]

Fotografia ta, jak i lektura licznie dostępnej literatury na ten temat pozwalają uzmysłowić, jak wyglądała praca w pierwszych systemach CAD. Był to proces żmudny w porównaniu z dzisiaj dostępnymi programami, a szczególnie z urządzeniami zapewniającymi łatwą i wygodną interakcję projektanta z interfejsem graficznym (wcześniej wszelkiego rodzaju digititzery, obecnie mysz, pióro świetlne czy nawet ekran dotykowy). W następnych latach środowisko CAD rozwijało się dość intensywnie, osiągając znaczną dynamikę w tym zakresie w latach 80. XX wieku, przede wszystkim z uwagi na rozpowszechnienie komputerów osobistych. Cennym źródłem wiedzy jest publikacja opisująca historię CAD [11].

Jednym z produktów rozwoju systemów CAD był ikoniczny już program AutoCAD. Pierwotnie powstał on dla systemu DOS (rys. 1.3), natomiast relatywnie szybko przeniesiono go do systemów graficznych Windows i Mac OS.



Rys. 1.3. Interfejs programu AutoCAD w systemie DOS [15]

AutoCAD to obecnie wciąż najpopularniejszy i dość wszechstronny program stosowany w wielu branżach przemysłowych i nie tylko. Mimo że doczekał się on wielu pokoleń następców w postaci podobnych programów CAD, a nawet ich klonów, to jednakże wydaje się, że w swojej klasie jest nadal liderem. Od wielu lat ciekawą alternatywą jest praca w tym programie w chmurze, która pozwala na zdalny dostęp do tworzonych projektów i to niekoniecznie przy użyciu komputera. System AutoCAD w tym wydaniu jest uruchamiany na notebookach, tabletach czy nawet smartfonach [16].

System CAD dość szybko zagościł w projektowaniu budowlanym. Pomimo rozwoju innych koncepcji i wręcz filozofii projektowania i komputerowego modelowania obiektów budowlanych system CAD wciąż jest podstawowym środowiskiem wielobranżowego projektowania budowlanego. Choć kierunek rozwoju

w tym zakresie został wytyczony już dawno, o czym poniżej, to wydaje się, że programy CAD na długi jeszcze czas będą podstawowymi i uniwersalnymi aplikacjami wspomagającymi pracę inżyniera projektanta wielu branż, nie tylko związanych z budownictwem, architekturą czy instalacjami. Podstawową zaletą w tym zakresie jest uniwersalność i otwartość koncepcji CAD, zgodnie z którą można zaprojektować w zasadzie każdy dowolny element, niekoniecznie związany ściśle z inżynierią. Dlatego też system CAD wciąż znajduje nowych użytkowników, niekoniecznie związanych z inżynierią. Właśnie ta uniwersalność CAD pozwala na możliwość jego integracji z innymi środowiskami czy systemami komputerowego wspomagania procesu produkcji, o czym wspomniano powyżej, omawiając systemy typu CAD/CAM. Istotną technologią w tym zakresie jest przetwarzanie w chmurze obliczeniowej niewspółmiernie przyspieszające możliwości obróbki danych i wydatnie skracające czas pracy potrzebny na stworzenie dokumentacji technicznej [17]. System CAD/CAM znajduje również użytek w inżynierii budowlanej, gdzie od dłuższego już czasu stosowany jest zautomatyzowany cykl produkcyjny oparty o obróbkę NC. Elementy konstrukcji metalowych są przygotowywane za pomocą obrabiarek sterowanych numerycznie czy nawet wypalane laserowo, a zbrojenie elementów żelbetowych jest prefabrykowane w postaci wygiętych prętów gotowych do wbudowania. Oprogramowanie CAD stosowane w projektowaniu budowlanym jest wyposażone w funkcje charakterystyczne dla systemów CAM, takie jak np. możliwość przygotowania detali w formacie CNC.

1.3. SYSTEM BIM

Tak jak zasygnalizowano powyżej, kierunki rozwoju systemów CIM, a w szczególności CAD, zostały wytyczone już dawno. Nastąpiło zawężenie i specjalizacja poszczególnych programów oraz całych środowisk funkcjonujących w tym formacie. Obecnie firmy produkujące oprogramowanie inżynierskie w swojej ofercie mają nawet po kilkadziesiąt programów przewidzianych do wspomagania procesu projektowania i produkowania różnorakich produktów. Są to wysoko specjalistyczne środowiska, w których można przeprowadzić cały proces projektowania, produkcji, a także i analizy funkcjonowania projektowanych produktów.

Na tym polu relatywnie dawno pojawiła się koncepcja zmiany podejścia do procesu modelowania projektowanego produktu. W pierwszych krokach definiowano geometrycznie elementy dwuwymiarowe, przedstawiając je w płaskich planach jako tak naprawdę rzuty na płaszczyzny. Kolejnym krokiem było przejście na trójwymiar, czyli modelowanie graficzne 3D. To dotyczyło nie tylko branży inżynierskiej, ale także, a może w pewnym zakresie przed wszystkim, grafiki 3D, związanej z modelowaniem i wizualizacją opartą na renderingu. Umożliwiło to zastosowanie wysokospecjalistycznych programów w branżach graficznej, filmowej czy wreszcie gier komputerowych. Jako ciekawostkę wystarczy tu wspomnieć pionierskie zastosowania programów komputerowych przez Tomasza Beksińskiego do tworzenia grafik i obrazów, któremu jak na ówczesne czasy udało się uzyskać nie-

samowite i profesjonalne efekty [18]. Od jakiegoś już czasu koncepcja CAD oparta na grafice 2D i 3D niejako się wypaliła, bo programy osiągnęły swą funkcjonalność na praktycznie maksymalnym poziomie i trudno w tym zakresie o jakieś innowacje. Tak jak wspomniano, poszukiwano nowej drogi wspomagania projektowania inżynierskiego. Wymagało to zmiany podejścia do tradycyjnego myślenia o procesie modelowania projektowanego produktu. Choć środowiska CAD były dopracowane w wysokim zakresie i oferowały szeroką funkcjonalność, to jednak sam proces modelowania oparty na grafice dwu- i trójwymiarowej był mało inteligentny i elastyczny. Wspomniana nowa droga wymagała zmiany filozofii projektowania, a tak naprawdę budowy modelu projektowanego produktu. W omawianym przypadku tym produktem finalnie jest obiekt budowlany. Wymyślono więc, że można przede wszystkim w modelu projektowanego obiektu zakodować pewne informacje wykraczające poza jego geometrię czy inne dane, które definiowane były do tej pory w ograniczonym zakresie. Tym samym model obiektu zacząłby pełnić w pewnym sensie funkcję bazy danych, gdzie kodowano by szereg informacji go definiujących. Druga innowacja to sam sposób budowy modelu. Rysowanie poszczególnych elementów składowych obiektu budowlanego nawet w oparciu o grafikę 3D w sposób niejako ręczny jest wysoce pracochłonne i czasochłonne, a tym samym nieefektywne. Zwyczajna praktyka była taka, że projekt danego elementu konstrukcyjnego to niejako przeniesienie tradycyjnego układu rysunków złożeniowych, wykonawczych i warsztatowych do przestrzeni programu, najczęściej w postaci rzutów, przekrojów i widoków 2D. Wyższy poziom to definicja projektowanych elementów jako obiektów trójwymiarowych. Do tej pory realizowane to było w zakresie ograniczonym dla danej branży, np. przy wykonywaniu dokumentacji wykonawczo-warsztatowej elementów konstrukcji metalowych. Podstawowym ograniczeniem był brak jednego uniwersalnego środowiska CAD, w którym można by było modelować dowolny obiekt budowlany jako obiekt w pełni trójwymiarowy i zdefiniowany z uwzględnieniem jego wszystkich specyficznych cech geometrycznych, takich jak np. elementy złączne (śruby, spoiny), zbrojenie elementów żelbetowych czy inne. Takie podejście do pewnego momentu było całkiem funkcjonalne, jednakże nową ideą było wymyślenie takiego sposobu definicji elementu, aby z jednej strony sam proces był łatwy, szybki i precyzyjny, a z drugiej osiągnięta zostałaby szeroka uniwersalność w zakresie typów modelowanych elementów. Wymyślono więc, że poszczególne elementy składowe obiektu budowlanego modelowane będą od razu w postaci obiektów trójwymiarowych odpowiadających geometrii danego elementu, a więc będą miały adekwatny przekrój, długość i charakterystyczne atrybuty geometryczne (np. otwory, podcięcia itp.) oraz będą definiowane w sposób uproszczony. To uproszczenie polega na prostym zdefiniowaniu w przestrzeni roboczej modelu charakterystycznych punktów modelowanego obiektu, np. początku i końca, oraz wczytaniu gotowego obiektu z bazy danych, będącego modelem projektowanego elementu. Model ten zawiera także szereg informacji definiujących inne poza geometrycznymi parametry

elementu, co jest pierwszą nowatorską funkcjonalnością nowej idei opisanej wcześniej. Ta nowa koncepcja modelowania przybrała postać tzw. technologii BIM (ang. *Building Information Modeling*) określanej jako modelowanie informacji o budynku, choć tak naprawdę obejmuje ona modelowanie całego procesu realizacji inwestycji budowlanej i zarządzanie nim.

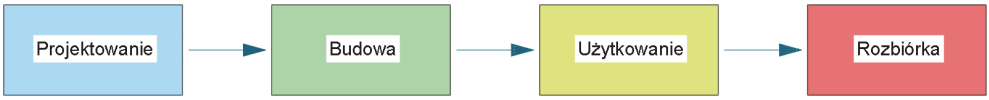
Koncepcja BIM kiełkowała w umysłach osób zajmujących się komputeryzacją procesu projektowania dość dawno, bo już w roku 1975 Chuck Eastman z Georgia Institute of Technology w USA sformułował pojęcie *Building Description System* (BDS) [19]. Zgodnie z jego ideą podstawą całego systemu powinien być przestrzenny, inteligentny model projektowanego obiektu. Umożliwiałby on przygotowanie w łatwy i zautomatyzowany sposób odpowiednich rzutów, przekrojów czy wizualizacji, bez konieczności generowania ich w sposób niezależny od modelu, jak ma to miejsce w tradycyjnym podejściu CAD 2D. W modelu BDS zakodowana byłaby informacja o najistotniejszych jego elementach składowych. W przytaczanym artykule autor zaprezentował szczegółową koncepcję tworzenia inteligentnych, składowych obiektów geometrycznych, z których tworzony byłby cały model. To nic innego jak obecnie stosowane sparametryzowane elementy modelu BIM. Informacje zakodowane w modelu BDS powinny zawierać nie tylko dane geometryczne, a projektant powinien w łatwy sposób móc uzyskać i wygenerować raporty dotyczące materiałów czy kosztów. Równie istotne było także to, aby informacja o inwestycji była zintegrowana i dostępna na wielu etapach jej realizacji.

W kolejnych latach dopracowywano i urzeczywistniano tę ideę, doprecyzowując nazwy nowej technologii. Na przykład w USA na początku lat 80. XX wieku wprowadzono nazwę *Building Product Models*, a w Europie, przede wszystkim w Finlandii, *Product Information Models* [20]. Co równie istotne, w okresie tym położono podwaliny w zakresie definicji i organizacji samego modelu obiektu budowanego, opracowując również oprogramowanie i formaty plików. Termin *Building Modeling* został zastosowany w roku 1986 przez Roberta Aisha w tytule pracy [21]. Koncepcję w niej przedstawioną można traktować jako wytyczne, fundamenty, na których oparto i zbudowano istniejącą dziś technologię BIM. Obejmowała ona spójny model trójwymiarowy, zawierający komponenty sparametryzowane i zakodowane w bazie danych. Sam proces tworzenia dokumentacji technicznej cechuje się wysokim poziomem automatyzacji. Dostępne są inne opcje, takie jak choćby tworzenie i sterowanie harmonogramem realizacji inwestycji. Termin *Building Modeling* z kolei został niejako rozwinięty przez G.A. van Nederveena i F.P. Tolmana w publikacji z roku 1992 [22]. Finalnie termin *Building Information Modeling* i pochodzący od niego akronim BIM jest autorstwa Jerry'ego Laiserina, który użył go w publikacjach [23, 24].

Wizja takiego podejścia do projektowania, a tak naprawdę do realizacji inwestycji i zarządzania nią aż do etapu rozbiórki obiektu dość szybko zaczęła się materializować. Koniecznością było opracowanie całej koncepcji systemu BIM, która jest niezbędna w tak kompleksowym podejściu do inwestycji budowlanej. Dotyczy

to nie tylko infrastruktury informatycznej w postaci środowisk modelowania komputerowego, ale również, a może przede wszystkim samego systemu zarządzania informacją na wielu polach i etapach. Z tego powodu zdecydowano się na standaryzację całego procesu BIM. Następuje ona sukcesywnie w wielu krajach świata, natomiast podstawowe są normatywy międzynarodowe [25-35], które są na bieżąco aktualizowane. Stanowią one podstawowe zbiory reguł i procedur stosowania technologii BIM. Dla użytkowników wysoce pomocne są podręczniki, gdzie w przystępny sposób przedstawiana jest tematyka BIM. Fundamentalne jest tutaj pierwsze wydanie podręcznika autorstwa Chucka Eastmana i innych [20], jak i najnowsze [36]. Należy także wyróżnić krajowe pozycje [37, 38].

Obecnie terminem BIM określa się tak naprawdę proces cyfrowego modelowania całego cyklu życia obiektu budowlanego, a więc od jego najwcześniejszego okresu, czyli prac koncepcyjnych, przez zasadnicze projektowanie, wykonawstwo, użytkowanie, aż do rozbiórki (rys. 1.4).

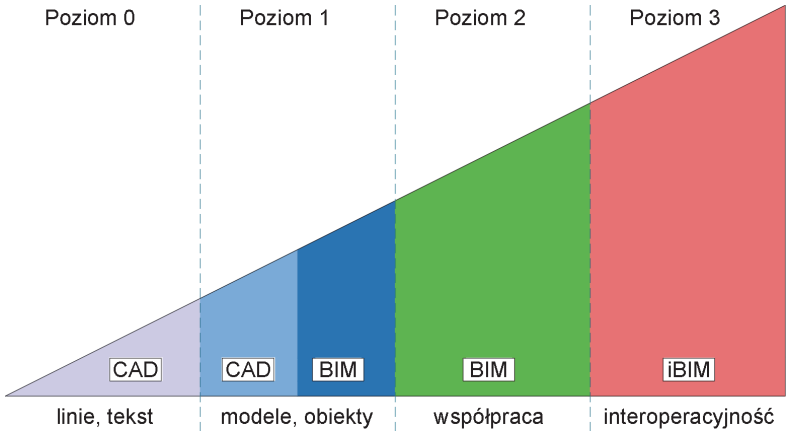


Rys. 1.4. Cykl życia obiektu budowlanego

Lepszym określeniem byłoby *modelowanie informacji o obiekcie budowlanym* albo wręcz *o budowaniu* (co jest aktualnie również stosowane) z uwagi na szerszą pojemność terminu *obiekt budowlany* w porównaniu do *budynku*. BIM jest obecnie stosowany w zasadzie we wszystkich branżach architektury, budownictwa i instalacji, wykraczając daleko poza zakres projektowania i wznoszenia budynków. Obecnie wariacji nazwy BIM jest przynajmniej kilka, bo *modelowanie informacji* odnoszone jest do *budynku*, *budowli*, wspomnianego wielokrotnie *obiektu budowlanego*, a także procesu *budowania*. W zależności od danej branży wyróżnia się także kilka innych poddziałów BIM, przeznaczonych poszczególnym specjalnościom.

Ale czym obecnie jest BIM? Wszystko, co napisano powyżej, nie straciło na aktualności i przez wiele najbliższych lat, a może i dekad będzie nadal aktualne. Koncepcja BIM spełnia wszystkie opisane powyżej założenia i dążenia w zakresie rozwinięcia filozofii modelowania produktu, jakim jest obiekt budowlany, a tak naprawdę procesu jego realizacji. BIM to nic innego jak cyfrowy i inteligentny zapis danych definiujących projektowany obiekt w zakresie jego cech fizycznych i funkcjonalnych. Inteligencja tego zapisu polega na opisie parametrycznym, opartym na podejściu bazodanowym. Jedną z podstawowych funkcjonalności jest dostęp do tych danych (modelu BIM) przez wszystkich uczestników procesu realizacji inwestycji na każdym jego etapie, a więc od fazy koncepcyjnej, projektowej, realizacji, użytkowania, aż po rozbiórkę obiektu.

W zależności od klasy informacji zakodowanych w modelu wyróżnia się kilka tzw. poziomów BIM pokazanych schematycznie na rysunku 1.5.

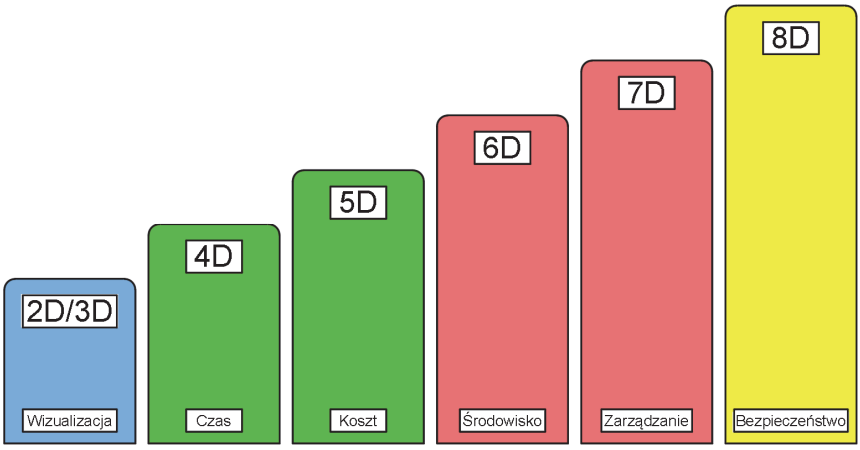


Rys. 1.5. Poziomy BIM

Jak widać, tradycyjny już system CAD to poziom zerowy, z uwagi na jedynie cyfrowe odwzorowanie klasycznej dokumentacji rysunkowo-technicznej. Współpraca BIM określana jako tzw. *integration and collaboration and interoperability BIM* na tym poziomie jest niewielka lub praktycznie w ogóle jej nie ma. Środki, za pomocą których odwzorowywana jest dokumentacja rysunkowa, to linie, łuki, tekst, a więc najprostsze z możliwych elementarnych (prymitywnych) elementów graficznych. W kolejnych poziomach klasyfikuje się coraz to bardziej zaawansowany sposób modelowania projektowanych obiektów, gdzie istnieje już możliwość operowania grafiką 2D i 3D, a poszczególne elementy składowe są modelowane jako obiekty inteligentne. Do ich definicji używane są swego rodzaju bloki, gdzie zakodowane są informacje składowe o geometrii, materiale i cechach specyficznych. Dopiero na poziomie drugim dochodzi do większej współpracy i wymiany danych oraz stosowania formatów pozwalających na definiowanie i kodowanie informacji zgodnie z omówioną powyżej koncepcją BIM. Na tym poziomie wyróżnia się również niejako podsystemy BIM przeznaczone specjalnie dla poszczególnych branż. Są to m.in. modelowanie informacji o konstrukcji, czyli SIM (*Structure Information Model*); model informacyjny architektury AIM (*Architecture Information Model*) czy model BIM dla obiektów mostowych BrIM (*Bridge Information Model*). Podsystemów tych jest oczywiście więcej i można się spodziewać, że w miarę opracowywania technologii BIM, rozbudowywania istniejących środowisk modelowania o coraz to większą liczbę branż będą one rozszerzane. Aktualnie najwyższy, trzeci poziom BIM to poziom, gdzie dochodzi do szerokiej i pełnej wymiany informacji i współpracy nie tylko międzybranżowej, ale współpracy wszystkich uczestników procesu inwestycyjnego i eksploatacji obiektu. Poziom ten określany jest jako iBM (ang. *interoperable Building Information Model*), czyli interoperacyjny cyfrowy model obiektu. Fundamentalną rolę w wymianie interoperacyjnej spełnia możliwość transferu informacji za pomocą otwartych formatów modeli, takich jak obecnie podstawowy format IFC.

Osobną kwestią jest tzw. wymiarowość technologii BIM. Określa ona nie tylko ilość wymiarów w sensie geometrycznym, ale kolejne funkcjonalności, które model w danym wymiarze obejmuje. Wymiary BIM są określane najczęściej jako xD, gdzie litera x oznacza kolejny numer. Klasyczna dokumentacja techniczna to wymiary 2D i 3D BIM, gdzie podejście 2D dotyczy rysunków przygotowanych cyfrowo w układzie płaskim, a 3D obejmuje najczęściej wizualizacje koncepcyjne czy renderingi, a także widoki trójwymiarowe konstrukcji. BIM 4D to model zawierający informacje i funkcjonalności w zakresie harmonogramowania, a więc dodatkowym parametrem poza geometrią jest czas. Kolejny wymiar 5D pozwala na ujęcie kwestii kosztowych, także w aspekcie czasowym, a więc umożliwia zautomatyzowane kosztorysowanie i budżetowanie realizowanej inwestycji. Analizy środowiskowe związane z koncepcją zrównoważonego rozwoju to funkcjonalność zawarta w modelu 6D, natomiast na zarządzanie gotowym obiektem pozwala wymiar 7D [38, 39]. Do tej pory ostatni wymiar BIM to 8D, gdzie w modelu zawarto informacje w zakresie bezpieczeństwa. Należy również zaznaczyć, że zakres niektórych wymiarów BIM przyporządkowywany bywał nieco inaczej, niż to podano powyżej, bo np. 6D oferował zarządzanie obiektem [40].

Poziomy i wymiary BIM są ze sobą związane, bo to właśnie rozwijane funkcjonalności tej technologii w powiązaniu z interoperacyjnością klasyfikują dany model, co pokazano schematycznie na rysunku 1.6.

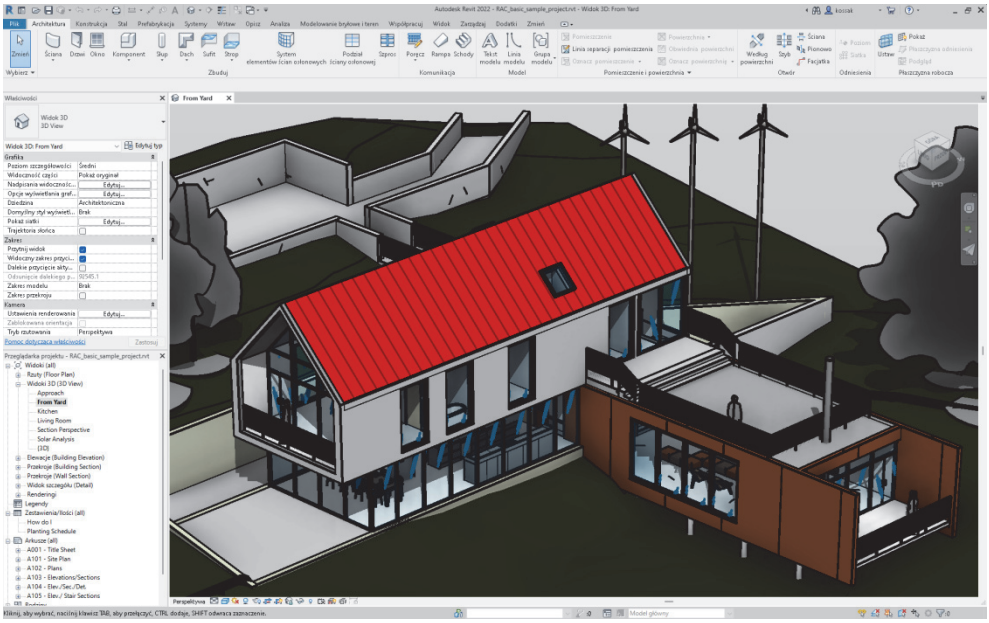


Rys. 1.6. Rozwój i wymiary BIM

Funkcjonalność BIM, jak i rozwój oprogramowania spowodowały trend dynamicznego upowszechnienia tej technologii. Skutkowało to m.in. wymogiem obligatoryjnego realizowania niektórych inwestycji w standardzie BIM [41], gdzie dokumentacja projektowa musi być zgodna m.in. z otwartym formatem BIM, np. IFC, a całość procesu jest zgodna z procedurami szczegółowymi.

Bardzo szybko technologią BIM zainteresowali się główni gracze na rynku CAD. Potentaci tacy jak Autodesk, Bentley, Trimble, Nemetschek czy Dlubal od

wielu już lat rozwijają swoje środowiska modelowania, a także wspomagania realizacji całego procesu inwestycyjnego zgodnie z formułą BIM. Jednym z najpopularniejszych jest Autodesk Revit pokazany na rysunku 1.7.



Rys. 1.7. Środowisko Autodesk Revit w wersji 2022

Rozwój oprogramowania to jedna ze ścieżek rozwoju technologii BIM. Bardzo istotna jest tutaj integracja międzybranżowa i interoperacyjność, a także rozszerzanie funkcjonalności pozwalającej na pracę na jednym modelu centralnym umożliwiającym modelowanie, obliczenia, optymalizację i detalowanie [42]. Bardzo ważne jest również dostosowanie środowisk projektowych do możliwości realizacji inwestycji branżowych w pełni zgodnie ze standardem BIM. Ma to już miejsce np. w branżach drogowej [43] czy mostowej, ale wiele branż jest w tym zakresie słabo zagospodarowanych, np. instalacje elektryczne.

1.4. SYSTEMY PROJEKTOWANIA SPARAMETRYZOWANEGO

Szeroko traktowana technologia komputerowa stosowana do wspomagania procesu projektowego zaowocowała powstaniem nie tylko przedstawionych powyżej systemów. Istotną i bardzo interesującą ścieżką jest powstanie i rozwój wielu specyficznych metod projektowania opartych właśnie na zastosowaniu komputeryzacji. Idąc dalej, należy wyróżnić idee i koncepcje architektoniczne, które są związane i oparte na wykorzystaniu tych systemów. Stosując podział przyjęty w [44], można wyróżnić kilka rodzajów projektowania opartego na zastosowaniu i wyko-

rzystaniu komputerów i metody numerycznych. Najistotniejsze z uwagi na tematykę niniejszej publikacji są następujące koncepcje i typy projektowania:

- projektowanie obliczeniowe (*Computational Design*), w najszerszy sposób traktujące proces projektowy obejmujący zarówno wykorzystanie potencjału obliczeniowego stosowanych urządzeń i metod, jak i samo projektowanie produktów i symulowanie procesów w oparciu o programy komputerowe;
- projektowanie algorytmiczne (*Algoritmic Design*), gdzie proces projektowania jest przeprowadzany za pomocą specjalnie przygotowanych do tego celu algorytmów komputerowych. Istotą jest tutaj możliwość przeprowadzania operacji na zbiorach danych wejściowych o szerokiej i zróżnicowanej naturze w celu uzyskania określonego efektu, np. w postaci geometrii projektowanego obiektu;
- projektowanie parametryczne (*Parametric Design*), w którym proces projektowania czy projektowanego produktu opisany jest sparametryzowanymi regułami. Projektant ma możliwość elastycznego manipulowania parametrami w celu uzyskania określonego celu;
- projektowanie generatywne (*Generative Design*) jest niejako pochodną i kolejnym etapem ewolucji opisanych powyżej metod i zawiera w sobie możliwość uzyskania wielowariantowości prowadzącej do optymalizacji projektowanego produktu w oparciu o zastosowanie inteligentnych algorytmów numerycznych. Proces ten oparty jest na następujących elementach takich jak warunki i parametry początkowe definiujące zadanie, warunki ograniczające, mechanizm (algorytm) sterujący i wybór wariantu optymalnego w oparciu o wachlarz przygotowanych (wygenerowanych) wariantów.

Istnieją oczywiście jeszcze inne rodzaje (ścieżki) procesów projektowych opartych na zastosowaniu metod i narzędzi komputerowych, jak również pewnego rodzaju hybrydy typów opisanych powyżej. Dotyczy to przede wszystkim projektowania parametryczno-algorytmicznego, które najlepiej chyba oddaje istotę zarówno projektowania parametrycznego, jak i algorytmicznego z uwagi na płynne ich przenikanie i koegzystencję [44].

W tym miejscu należy zauważyć mnogość dróg poszukiwania źródeł kształtowania form projektowanych obiektów, nie tylko architektonicznych, i w efekcie odnalezienia algorytmów je opisujących. Bardzo ciekawa jest lektura dostępnej w tym zakresie literatury [45-55]. Na szczególną uwagę zasługują także krajowe pozycje [44] i [56].

Tematyka podjęta w niniejszej monografii jest ściśle związana właśnie z projektowaniem algorytmiczno-parametrycznym, a dokładniej z wykorzystaniem istniejących obecnie narzędzi w postaci środowisk programistycznych w projektowaniu inżynierskim.

1.5. OPCJE PROGRAMISTYCZNE W ŚRODOWISKACH CAD/BIM

Obecnie funkcjonujące programy, czy to klasyczne CAD, czy BIM poza swoim podstawowym przeznaczeniem i funkcjonalnością, w przypadku CAD cyfrowym opisem projektowanego produktu, w BIM modelowaniem i kodowaniem kompleksowej informacji o obiekcie budowlanym, są wyposażane w szereg dodatkowych opcji i rozszerzeń. Przede wszystkim środowiska te (z przewagą BIM) są zintegrowane z systemami i programami stosowanymi w technologii całościowego komputerowo zintegrowanego wytwarzania, czyli *Computer Integrated Manufacturing*.

Jedną z funkcjonalności stanowiących niejako wartość dodaną wyżej wymienionych środowisk jest możliwość wykorzystania programowania do generacji czy to cyfrowego obrazu dokumentacji technicznej CAD, czy to modelu BIM. To bardzo istotna opcja, ponieważ pozwala na o wiele większą swobodę w kształtowaniu danego modelu, jak również w jego edycji, zarządzaniu nim, czy choćby przygotowaniu do wysłania do środowisk zewnętrznych. Stosowane są tu różne środowiska programistyczne i języki programowania. Niebagatelna jest kwestia parametryzacji opisu modelu, niekoniecznie dotycząca tylko jego geometrii. Jedną z takich opcji jest programowanie wizualne, któremu poświęcona jest niniejsza monografia. Poniżej przedstawiono ogólną koncepcję programowania opartą właśnie na kodzie wizualnym, jak również gotowe środowiska programistyczne, w tym środowiska wykorzystywane w projektowaniu inżynierskim i modelowaniu 3D.

Od zarania dziejów projektowanie było procesem wizualizowania koncepcji twórcy, archaicznego projektanta, który na przestrzeni wieków zaczął tworzyć pierwsze rysunki techniczne, układające się w zarys dokumentacji technicznej. Nie sposób w tym miejscu nie wspomnieć o pierwszym zachowanym traktacie na temat architektury autorstwa Witruwiusza pt. *O architekturze ksiąg dziesięć* [57]. W naszych czasach działania te przybierają formę projektu, gdzie są przedstawiane poszczególne elementy składowe obiektu. Tak naprawdę wizualizacja tych elementów w zakresie dokumentacji projektowej to nic innego jak przedstawienie projektowanych części obiektu budowlanego za pomocą elementów składowych, przywoływanych już tzw. *primitives*. Są to linie, krzywe, figury i bryły geometryczne, mniej lub bardziej foremne. Projekt to nic innego jak model obiektu budowlanego składający się z tego typu elementów geometrycznych. Jest to więc zbiór elementów i figur matematycznych. Jak wiadomo, obiekty te można narysować wprost na kartce papieru czy wprowadzić w sposób graficzny w przestrzeni modelu komputerowego. Można tego jednak dokonać w sposób bardziej usystematyzowany, posługując się opisem matematycznym. Zatem każdy z definiowanych elementów musi zostać dokładnie umiejscowiony w przestrzeni określonej danym układem współrzędnych, niekoniecznie kartezjańskich *xyz* i opisany daną funkcją. Dzięki takiemu podejściu można manipulować modelowanym elementem, sterując jego parametrami. W odniesieniu do linii byłyby to np. współczynniki kierunkowe funkcji ją opisującej czy granice określające jej argumenty i wartości. To w matematyczny sposób determinuje punkty początku i końca linii, a więc precyzyjnie określa poło-

żenie linii w układzie *XYZ*. Taki sposób definiowania poszczególnych elementów składowych modelujących np. konstrukcję projektowanego obiektu byłby bardzo żmudny i nieefektywny. Jest to jednak realizowane np. w systemach CAD w sposób niejako ukryty przed użytkownikiem, bo przecież modele tworzone w tym środowisku są ściśle, matematycznie zdefiniowane i umiejscowione w przestrzeni modelu projektu.

Koncepcja precyzyjnego lokalizowania każdego elementu w przestrzeni w sposób zdeterminowany jego jawnymi współrzędnymi, jak to ma miejsce np. w programach typu AutoCAD, została zarzucona na rzecz definiowania nieco niejawnego, jak np. w środowisku BIM Revit. Użytkownik funkcjonuje w przestrzeni modelu, ale nie opiera się na jawnym układzie odniesienia, w postaci np. kartezjańskiego układu współrzędnych, natomiast poszczególne elementy są definiowane przede wszystkim w sposób relatywny. Są one lokalizowane w oparciu o pewne elementy referencyjne w postaci choćby tzw. poziomów i siatki osi. Koncepcja zastosowana w systemie Revit jest więc pewnym krokiem w kierunku usprawnienia samego procesu definiowania poszczególnych elementów modelu. Projektant nie musi nadmiernie pilnować dokładności podczas wstawiania poszczególnych elementów, bo system mu w tym pomoże, dopasowując newralgiczne punkty wprowadzanych elementów do innych obiektów istniejących. Jest to wiec podejście intuicyjne, gdzie system w inteligentny sposób niejako kieruje procesem definicji poszczególnych elementów składowych modelu. Jednakże cały czas jest on zbiorem mniej lub bardziej inteligentnych elementów składowych, które są zbiorem podstawowych figur geometrycznych. Kluczową sprawą jest tutaj sposób ich definicji, opisu i sposobu wprowadzania do modelu obiektu.

W tym miejscu można przejść do momentu kluczowego z punktu widzenia zmiany filozofii myślenia o procesie definicji modelu projektowanego obiektu. A co gdyby jednak uprzeć się, aby model definiować w sposób ściśle zmatematyzowany, a więc opisany w oparciu o definicję poszczególnych elementów za pomocą „parametrów" matematycznych? Każdy element byłby precyzyjnie opisany, a posiadając reguły opisu, można by go dowolnie modyfikować. Podstawowym problemem jest tutaj sposób opisu definiowanych elementów. Trudno sobie wyobrazić środowisko matematyczne, w którym funkcjonowałaby baza reguł determinujących w sposób ciągły (funkcyjny) lub zdyskredytowany (jako zbiór mniejszych składników tworzących cały element), za pomocą której można by tworzyć model projektowanego produktu stanowiący podstawę do wytworzenia dokumentacji technicznej w postaci projektu. Inną możliwością byłoby stworzenie opisu modelu w postaci składni, w której zakodowane byłyby procedury stanowiące program generujący taki model. Dopiero tego typu podejście dałoby szansę na stworzenie filozofii modelowania opartej na parametrycznym opisie modelu. Dotychczas nie wymyślono nic innego w tym obszarze, dlatego należało zaprojektować lub zaadaptować istniejące rozwiązania programistyczne do tego, aby możliwe było w miarę proste definiowanie modelu produktu oparte na podejściu parametrycz-

nym. Do rozwiązania tego problemu potrzebne było stworzenie środowiska lub wykorzystanie istniejących narzędzi, które spełniałyby uwarunkowania utylitarne, tzn. były przyjazne dla użytkownika – projektanta, który niekoniecznie jest przecież wysokowyspecjalizowanym programistą. Jedną z koncepcji w tym zakresie było oprogramowanie istniejących środowisk graficznych, np. CAD, tak aby za pomocą nieskomplikowanych skryptów, czyli klasycznego kodowania, możliwe było definiowanie i modyfikacja obiektów graficznych należących do biblioteki danego programu. Przykładem może tutaj być język programowania AutoLisp, funkcjonujący w programie AutoCAD. Dzięki temu możliwa była automatyzacja procesu projektowania w tym programie, jak również integracja z innymi aplikacjami. Drugi kierunek to poszukiwanie sposobu na programowanie procesu modelowania, ale w oparciu o inne niż klasyczne metody kodowania. Do tego celu zdecydowano się na wykorzystanie sposobu kodowania opartego na programowaniu nietekstowym. Zanim przedstawiona zostanie ta druga koncepcja, należy dokonać pewnego wprowadzenia natury psychologicznej.

Jak wiadomo, człowiek jest wyposażony w kilka zmysłów, z których jednym jest zmysł wzroku. Oprócz swojej podstawowej funkcji, tj. odbioru bodźców świetlnych ze świata i zobrazowania go w świadomości, zmysł wzroku odgrywa fundamentalną rolę w szeroko rozumianym poznawaniu świata, niekoniecznie tylko materialnego. Człowiek nie ma pełnej, a często żadnej świadomości, ile informacji pobiera i magazynuje w pamięci za pomocą zmysłu wzroku. Wydaje się, że do tej pory nie wykorzystuje się efektywnie możliwości poznania wizualnego w procesie edukacyjnym. Obrazowanie problemów, sytuacji będących przedmiotem choćby zadań matematycznych pozwala lepiej wyobrazić sobie opisywaną sytuację, co wymiernie pomaga w znalezieniu rozwiązania, które może być z kolei przedstawione za pomocą operacji i formuł matematycznych. Innym przykładem na efektywne wykorzystanie poznania wizualnego może być tzw. czytanie nut, gdzie przed ich próbą odtworzenia na instrumencie muzyk analizuje przebieg linii melodycznej, poszczególnych fraz czy struktury harmonicznej. Osoba odpowiednio wykształcona muzycznie jest niejako wytrenowana w zakresie rozumienia zapisu nutowego, myślowego przetworzenia go i odtworzenia za pomocą narządu ruchowego, zależnego oczywiście od instrumentu. Osobną kategorią są śpiewacy, którzy często potrafią bezbłędnie odtworzyć dany utwór *a capella* i *a vista*.

Mechanizm poznania wizualnego jest dość skomplikowany i stanowi wciąż obiekt intensywnych badań psychologicznych [58]. Kluczową kwestią jest rozpoznawanie kształtów i kolorów, co powoduje rekognicję, czyli ustalenie i potwierdzenie tożsamości przedmiotu, osoby czy rzeczy. Mechanizm ten jest od dłuższego czasu wykorzystywany praktycznie, choćby w reklamie, gdzie, aby zainteresować potencjalnego klienta danym produktem, niejako bombarduje się go informacją na jego temat. W reklamie wizualnej podstawowym elementem jest właśnie jego obraz, często potęgowany efektami opartymi na kolorze. Sama koncepcja operowania kształtami czy elementami geometrycznymi jest używana w wielu dziedzinach

i zagadnienie to mogłoby być tematem osobnej rozprawy. Najistotniejsze dla omawianego w monografii zagadnienia są możliwości zastosowania środowiska wizualnego w inżynierii, a konkretniej w programowaniu komputerowym, o czym traktuje poniższy podrozdział.

1.6. PROGRAMOWANIE WIZUALNE

W naturalny sposób wykorzystanie możliwości ludzkiego mózgu w zakresie programowania zostało m.in. ukierunkowane i oparte na zmyśle wzroku. O wiele łatwiej manipulować obiektami graficznymi w celu osiągnięcia danego celu, niż opisywać je za pomocą składni językowej czy równań matematycznych. Przecież jedną ze zdolności, jaka rozwija się u małych dzieci, jest właśnie manipulacja elementami przestrzennymi w formie brył tak, aby uzyskać pożądany efekt, np. przecisnąć sześcian przez otwór o przekroju kwadratowym, a nie kołowym. To samo tyczy się zabawy klockami, np. Lego, czego doświadczają już całe pokolenia, i to niekoniecznie dzieci. Jednym z efektów takiej zabawy jest wprowadzenie do programowania i robotyki.

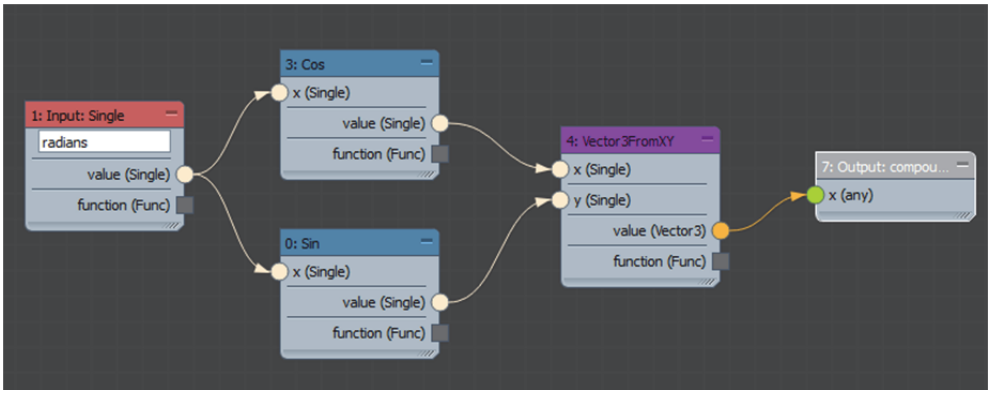
A gdyby zbudować pewien system obiektów graficznych, które po pierwsze, że posiadałyby pewne samoistne zdolności przeprowadzania operacji, to po drugie mogłyby ze sobą koegzystować i współpracować? Programista nie musiałby pisać żmudnych i w sumie nudnych sekwencji komend, które jak na złość od razu nie działają, tak jak należy i trzeba się nieźle natrudzić, żeby je uporządkować i zmusić tym samym do poprawnego działania. Taką ideę można wywnioskować, patrząc z pewnego dystansu na opracowane rozwiązania oparte właśnie na programowaniu nietekstowym, ale wykorzystującym inteligentne elementy graficzne, posiadające pewną zdolność. Objawia się ona możliwością wykonywania przez obiekty graficzne pewnych operacji na zbiorach danych – w blokach graficznych zakodowane są procedury-komendy pozwalające na przetwarzanie danych. Bloki te muszą tworzyć pewną większą całość, populację bloków, jeśli powierzono im jakieś większe zadanie do wykonania. W tym miejscu można przejść do programowania określanego w programistycznej nomenklaturze informatycznej jako *wizualne języki programowania*. Zgodnie z terminologią anglojęzyczną, czyli podstawową z punktu widzenia programowania, używany jest termin *Visual Programming Languages*, określany akronimem VPL. To nic innego, jak szeroka grupa języków programowania opartych na inteligentnych blokach, zdolnych do przetwarzania danych, które odpowiednio połączone ze sobą mogą te dane przekazywać do dalszego przetwarzania. W efekcie takiego powiązania użytkownik buduje algorytm stanowiący kod graficzny, który jest w pełni funkcjonalnym programem. Jedyna różnica jest taka, że w klasycznym kodowaniu standardowo programista pisze skrypt, a w oparciu o programowanie wizualne pojawia się twór, który przybiera postać struktury wzajemnie powiązanych ze sobą obiektów graficznych, będących odzwierciedleniem elementów składni danego języka programowania. Kod graficzny funkcjonujący w językach klas VPL określany jest także jako skrypt wizualny (*visual script*).

Obecnie istnieje bardzo dużo języków VPL, które znajdują zastosowanie w wielu dziedzinach, o czym będzie mowa w dalszej części monografii.

Korzyści z graficznego, wizualnego podejścia do programowania zostały dostrzeżone już dawno. Przede wszystkim języki VPL polecane są w edukacji jako pierwsze języki programowania. Ich nauczanie realizowane jest już od wielu lat również w Polsce. Dzieci posiadające przecież umiejętność logicznego myślenia i bardzo lubiące zabawę w „podstępny" niejako sposób są wprowadzane w świat kodowania, tworząc np. skrypty w języku VPL o nazwie Scratch.

1.7. IDEA VPL

Ogólna idea języków typu VPL polega na zastąpieniu składni tekstualnej obiektami graficznymi, w których są zakodowane poszczególne komendy lub zawarte dane, oraz utworzeniu powiązań pomiędzy nimi w sposób graficzny. Tym samym podstawowymi elementami języka VPL, ale i środowiska programistycznego są bloki komend i elementy je łączące. Przyjmują one odpowiednią formę, np. tzw. węzłów (*nodes*) i łączników (*wires*), które stanowią strukturę skryptu i odpowiadają za obróbkę i przetwarzanie danych oraz ich przepływ (*data flow*). Na rysunku 1.8 pokazano przykład skryptu wizualnego przygotowanego w środowisku Max Creation Graph, które jest pewnym zawężeniem w stosunku do tradycyjnych języków VPL. Zaprojektowano je do obróbki danych w oparciu o funkcje matematyczne i jest ono określane jako język typu FVPL (ang. *Functional Visual Programming Language*).



Rys. 1.8. Budowa wizualnego języka programowania na przykładzie kodu środowiska Max Creation Graph (MCD) [59]

Jak widać, przygotowanie powyższego programu w sposób wizualny jest bardzo proste, bo wystarczy połączyć ze sobą kilka węzłów, aby dokonać obróbki i przepływu danych oraz otrzymania żądanego wyniku.

Oczywiście języki VPL odzwierciedlają składnię tekstualną, którą można by przygotować w sposób tradycyjny, pisząc po prostu skrypt. Częstokroć jest to jed-

nakże proces dość żmudny, bo przede wszystkim z założenia przygotowanie skryptu wizualnego ma być proste, przejrzyste i szybkie, a sama składnia tekstualna danego języka VPL bywa bardzo skomplikowana. Przykładem tego niech będzie miniskrypt VPL w środowisku Dynamo, gdzie zadaniem jest wykonanie operacji mnożenia dwóch liczb. Na rysunku 1.9 pokazano skrypt wizualny, a na rysunku 1.10 fragment zapisu tego skryptu w języku DesignScript, który jest tekstualnym językiem (*silnikiem*) programowania tego właśnie środowiska.

Rys. 1.9. Skrypt wizualny środowiska Dynamo

```
{
  "Uuid": "364120d9-665f-4410-8306-f5518ec164cf",
  "IsCustomNode": false,
  "Description": null,
  "Name": "1.X mnozenie",
  "ElementResolver": {
    "ResolutionMap": {}
  },
  "Inputs": [],
  "Outputs": [],
  "Nodes": [
    {
      "ConcreteType": "CoreNodeModels.Input.DoubleInput, CoreNodeModels",
      "NodeType": "NumberInputNode",
      "NumberType": "Double",
      "InputValue": 2.0,
      "Id": "1c14c3f0cb0544b3be1cd7bc2050d62c",
      "Inputs": [],
      "Outputs": [
        {
          "Id": "b19e4f85aa5640d5bd49cf939a1c643a",
          "Name": "",
          "Description": "Double",
          "UsingDefaultValue": false,
          "Level": 2,
          "UseLevels": false,
          "KeepListStructure": false
        }
      ],
      "Replication": "Disabled",
      "Description": "Tworzy numer."
    },
```

Rys. 1.10. Skrypt tekstowy DesignScript

Nie trzeba w tym przypadku komentować konfrontacji podejścia VPL do tradycyjnego kodowania. Zgodnie z ogólną ideą kodowanie wizualne jest o wiele efektywniejsze, jeśli chodzi o przygotowanie skryptu w porównaniu do programowania tekstualnego.

1.8. JĘZYKI VPL

Na początek należałoby określić, czym w zasadzie są języki klasy VPL. Zgodnie z definicją podaną w internetowym słowniku FolDoc *visual programming language (VPL) is any programming language that allows the user to specify a program in a two- (or more)-dimensionsional way* [60]. Jest to więc dowolny język programowania, który umożliwia użytkownikowi określenie programu w sposób dwu- lub więcej wymiarowy. W przypadku konwencjonalnych języków programowania ma miejsce zetknięcie się z przetwarzaniem jednowymiarowym, ponieważ kompilator lub interpreter przetwarzają strumienie znaków, które są jednowymiarowe. Takie rozróżnienie jest przedstawiane w innych źródłach, np. [61].

Istotną cechą podziałów czy klasyfikacji języków VPL jest ich ekspresja wizualna, w tym elementów graficznych stanowiących ich „wizualną składnię", takich jak ikony, formularze, diagramy itp. Należy także zaznaczyć, że niektóre języki używające środowisk graficznych nie są językami VPL, gdyż wykorzystują jedynie graficzny interfejs GUI ułatwiający programowanie.

Wizualne języki programowania zostały opracowane i wdrożone dość dawno, bo ich historia sięga kilkudziesięciu lat. Ich rozwój, ale może bardziej upowszechnienie jest w pewnym stopniu związane z rozwojem środowisk graficznych, które ściśle były związane z rozwijaniem systemów komputerowych, w tym opracowaniu systemów Windows. Wizualne języki programowania zostały wdrożone w wielu różnych dziedzinach. W oparciu o klasyfikację podaną w [62] można je podzielić na grupy obejmujące następujące zastosowania:

- edukację,
- multimedia,
- gry wideo,
- systemy/symulację,
- automatyzację,
- hurtownie danych,
- dziedzictwo,
- różne.

Sam system klasyfikacji języków VPL został zaproponowany również dość dawno, bo zaprezentowano go już w roku 1994 w publikacjach [63, 64]. Języki VPL są sukcesywnie rozwijane, wdrażane i upowszechniane, o czym świadczyć może rosnące nimi zainteresowanie w środowisku programistycznym i inżynierskim, jak również stworzenie w renomowanych wydawnictwach specjalnych czasopism, takich jak choćby wydawany przez Elsevier od roku 1990 "Journal of Visual Languages and Computing" [65] i powstały m.in. na jego bazie "Journal of Computer Languages" [66].

1.9. WYBRANE GRUPY JĘZYKÓW VPL

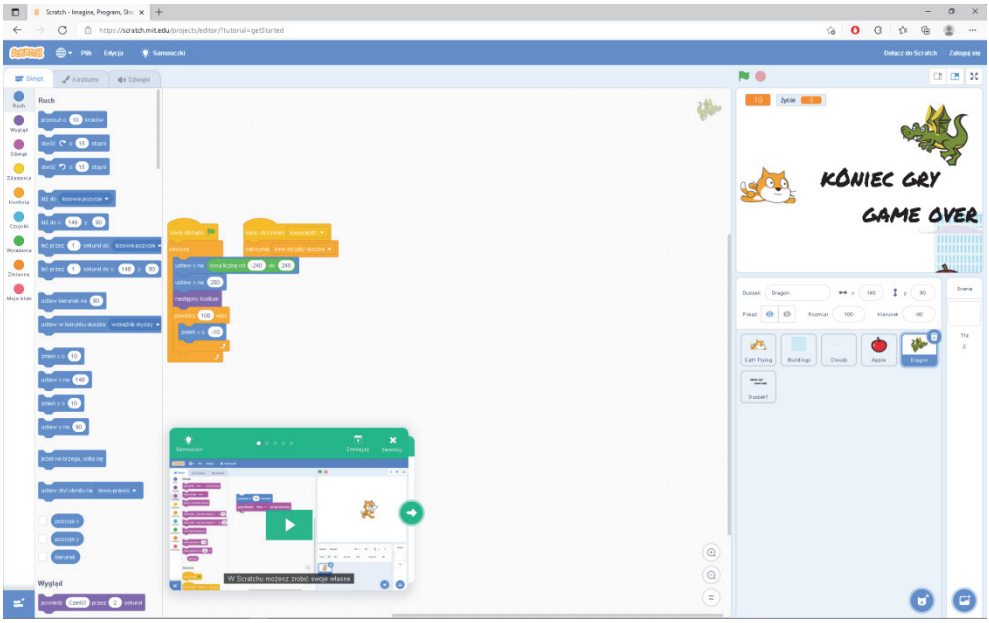
1.9.1. Języki edukacyjne

Kluczową kwestią przy omawianiu zagadnienia jakiegokolwiek programowania jest edukacja przyszłych programistów. Rzeczą fundamentalną w tym przypadku jest rozwijanie umiejętności logicznego myślenia i nauczania matematyki. O ile matematyka to, przynajmniej na razie, jeden z podstawowych przedmiotów nauczanych od szkoły podstawowej na całym świecie, to o tyle kwestia nauczania logiki najczęściej stanowi przedmiot edukacji na poziomie studiów wyższych i jest związana z wykształceniem np. filozoficznym.

Nauczanie tzw. informatyki w naszym kraju jest realizowane już od wczesnych klas szkoły podstawowej i ma na celu zapoznanie dzieci z ogółem zagadnień świata cyfrowego i stosowanych urządzeń oraz technologii IT. W starszych klasach wprowadzane są elementy programowania i co ciekawe są to właśnie języki wizualne. Ma to związek z przyjaznością środowiska i interfejsów tego typu programowania oraz względnej prostoty tych języków. Patrząc głębiej, kwestią podstawową są uwarunkowania psychologiczne, omawiane wcześniej, związane z poznawaniem wizualnym. Lista edukacyjnych języków programowania jest całkiem długa. Podając choćby za [67], są to języki Karel, Stagecast Creator, Lightbot, Kodu Game Lab, Scratch, Logomocja, RoboMind, Etoys, Alice, Lego Mindstorms, Mama. Języki te są rozwijane przez różnych producentów, nie wyłączając także aktualnych gigantów branży IT. Przykładowo projekt AppInventor był wspierany przez Google, przechodząc w produkt typu opensource, wspierany przez ogólną społeczność. Aktualnie jest on dostępy na stronie www.appinventor.mit.edu [68].

Edukacyjnym językiem wizualnym, który w naszym kraju jest powszechnie stosowany, jest język Scratch. Jest on nauczany od wczesnych klas szkoły podstawowej. Na jego przykładzie przedstawiona zostanie idea programowania wizualnego realizowanego w ramach edukacji szkolnej.

Oficjalna strona projektu Scratch to www.scratch.mit.edu [69]. Uczeń – programista korzysta z tego środowiska on-line, kodując bezpośrednio w interfejsie przeglądarki internetowej (korzystając z podanego wyżej wymienionego adresu) lub za pomocą wersji desktopowej udostępnianej przez Microsoft. Sam proces tworzenia skryptu wizualnego jest bardzo prosty, a interfejs graficzny przyjazny i kolorowy. Ta dodatkowa zachęta do nauki przez zabawę przyciąga coraz to młodsze pokolenia być może przyszłych programistów profesjonalnych. Za pomocą opcji *Stwórz* przechodzi się do interfejsu – środowiska graficznego, gdzie budowany jest skrypt wizualny. Po lewej stronie w kolumnie dostępne są odpowiednio pogrupowane komendy – procedury wykonywalne. Mają one wygląd kolorowych kafelków, które są przenoszone za pomocą myszki z biblioteki do przestrzeni roboczej skryptu (rys. 1.11). W niej programista odpowiednio zestawiając bloki – komendy, buduje skrypt wizualny. Jest on uruchamiany bezpośrednio w interfejsie za pomocą ikony zielonej flagi.



Rys. 1.11. Interfejs środowiska Scratch i przykładowy skrypt [70]

Język Scratch to produkt stanowiący pewnego rodzaju zachętę do dalszego programowania. Uczeń w nim programujący w formie choćby zabawy wyrabia w sobie umiejętność logicznego myślenia, na bieżąco obserwuje działanie i skutki poszczególnych komend, które ułożone w jednokierunkowym ciągu mogą spowodować skutek diametralnie odmienny od oczekiwanego. Można śmiało stwierdzić, że nauka programowania za pomocą języka Scratch to dobry kierunek, jeśli chodzi o naukę programowania na wczesnych etapach edukacji. Dodatkowo z uwagi na stopień zaawansowania i funkcjonalność język ten jest honorowany w konkursach kuratoryjnych z informatyki, co świadczy o jego wysokiej przydatności w omawianym zakresie. Istotny jest również fakt bogatej literatury, która jest udostępniana darmowo na stronie programu [69], a także w postaci podręczników, np. [71], i przykładów służących nauce. Na szczególne podkreślenie zasługuje również propagowanie i wprowadzanie tego języka do edukacji uczniów z dysfunkcjami, np. słuchowymi. Przydatne są w tym zakresie materiały dydaktyczne w postaci choćby publikacji metodycznej prowadzenia zajęć [72].

1.9.2. Języki profesjonalne

Zaprezentowany powyżej język Scratch, jak i inne wcześniej wymienione języki edukacyjne to tak naprawdę dopiero wstęp do prawdziwego kodowania wizualnego. Ta gałąź programowania jest cały czas rozwijana i choć niektóre z opracowanych środowisk i języków traktowane są czasami jako tzw. wstępniaki, czyli kody przygo-

towane do przetestowania danego zadania, to wydaje się, że wiele z nich może śmiało konkurować z czołowymi językami programowania stosowanymi profesjonalnie. Co istotne, wizualne języki programowania, tak jak wspomniano, nie wymagają szerokiej wiedzy programistycznej, dzięki czemu pozwalają one w krótkim czasie na nauczenie się i przygotowanie czasami całkiem efektywnie działającej procedury, wydatnie skracającej i automatyzującej rozwiązanie danego problemu.

Szeroka lista języków klasy VPL jest dostępna pod adresem [62] i z uwagi na jej obszerność nie ma sensu jej tu przytaczać. Jednakże nie jest to lista pełna i zamknięta, bo nie uwzględnia choćby języków stosowanych w projektowaniu inżynierskim, które traktowane są przez niektórych jako samodzielne środowiska programistyczno-designerskie. Krótki przegląd języków klasy VPL znajdujących właśnie takie zastosowania przedstawiono w kolejnym rozdziale.

2 ŚRODOWISKA OPARTE NA VPL STOSOWANE W PROJEKTOWANIU

Obecnie istnieje kilka środowisk programowania wizualnego, których podstawowy obszar działania jest wpisany w szeroko rozumiane komputerowe modelowanie obiektów budowlanych. Jego zakres jest bardzo szeroki, bo obejmuje podstawowe branże związane z projektowaniem, czyli architekturę, budownictwo i instalacje, ale z uwagi na otwartość przyjętej filozofii w zasadzie nie ma ograniczeń, aby zastosować je do innych bardziej specjalistycznych branż czy specjalności. Jest to sukcesywnie realizowane, bo obserwuje się zastosowanie programowania wizualnego także w innych obszarach. Tyczy się to również innych etapów realizacji inwestycji budowlanych niezwiązanych z projektowaniem, takich jak choćby fazy realizacji czy użytkowania. Tematyka niniejszej monografii dotyczy projektowania i w tym obszarze przedstawione zostaną informacje na temat systemów opartych na programowaniu wizualnym, które powstały jako pierwsze, jak również omówione zostaną środowiska używane aktualnie.

2.1. PROGRAMOWANIE W SYSTEMACH CAD

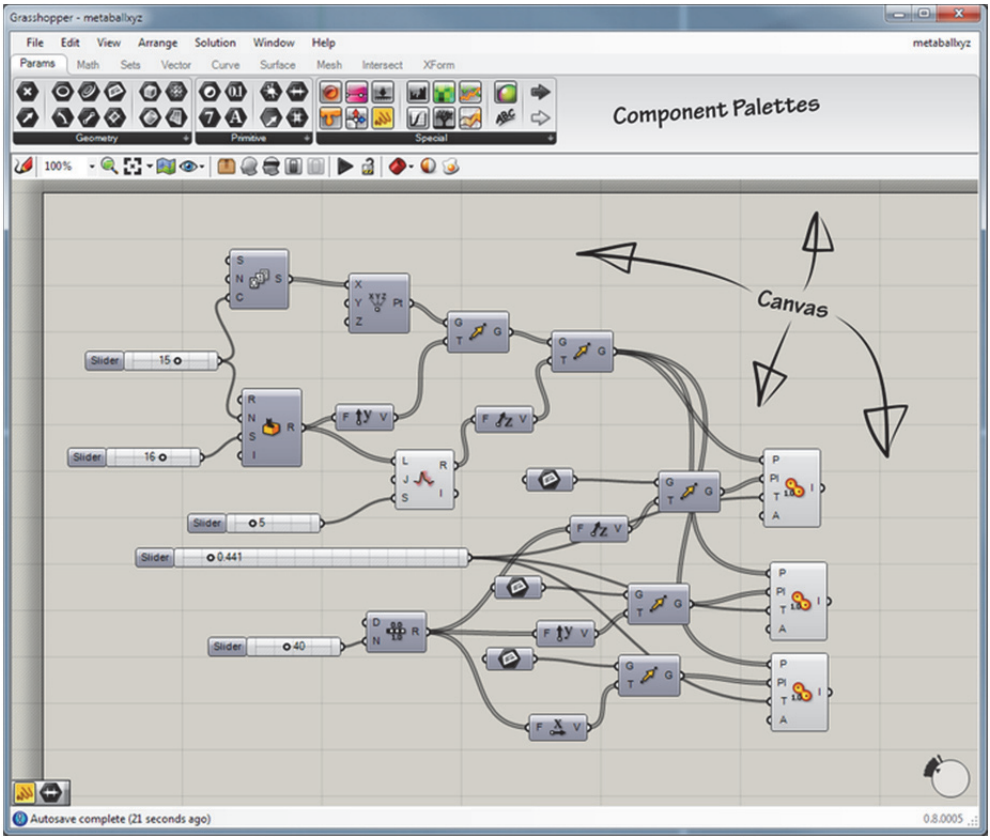
Podstawową funkcjonalnością, która umożliwiła zastosowanie programowania wizualnego jako narzędzia do wspomagania projektowania inżynierskiego, była możliwość tworzenia, kształtowania i optymalizacji grafiki 3D. Elementy, które tradycyjnie rysowane były w środowiskach CAD/CAM w przestrzeni 2D i 3D, dzięki podejściu sparametryzowanemu mogły być opisywane w postaci obiektów określonych matematycznie. To kluczowa kwestia z punktu widzenia całej filozofii. Tak naprawdę od początków programowania, w miarę coraz to większych możliwości graficznych za pomocą „tradycyjnego" kodowania tworzone były obiekty graficzne, które znajdowały zastosowanie w wielu dziedzinach, od inżynierii po zastosowania w branży multimedialnej (grafika, film itp.).

Pewnie mało kto dzisiaj pamięta, ale tego typu podejście prezentował język AutoLisp funkcjonujący w programie AutoCAD i to w wersjach działających w systemie DOS! Pisząc skrypty w tym języku, możliwa była definicja elementów graficznych funkcjonujących w środowisku AutoCAD, a także co najważniejsze również ich szybka i łatwa edycja. Choć język AutoLisp to nie język klasy VPL, to posiadał on wspomnianą już jedną z zalet programowania wizualnego – możliwość elastycznej, dynamicznej i przede wszystkim łatwej i szybkiej modyfikacji kształtowanej geometrii w środowisku projektowym, w tym wypadku CAD.

2.2. ŚRODOWISKA OPARTE NA PROGRAMOWANIU WIZUALNYM STOSOWANE W PROJEKTOWANIU INŻYNIERSKIM

2.2.1. Rhinoceros-Grasshopper

Jednym z pierwszych języków typu VPL, który został opracowany i wdrożony do wspomagania modelowania i projektowania inżynierskiego, był język Grasshopper (rys. 2.1). Pierwsza jego wersja, zwana wówczas Explicit History, została wydana we wrześniu 2007 roku. Język Grasshopper jest nierozerwalnie związany ze środowiskiem projektowym Rhinoceros, które funkcjonuje również pod nazwami Rhino lub Rhino3D. Całość to tak naprawdę środowisko programistyczne stanowiące program typu CAD/CAM/CAE.



Rys. 2.1. Jedno z wcześniejszych wydań środowiska Grasshopper [73]

Podstawową funkcją i przeznaczeniem środowiska Rhinoceros-Grasshopper jest tworzenie i modelowanie obiektów 3D. Jego zastosowanie jest szerokie, bo obejmuje przygotowanie modeli fizycznych produkowanych obiektów, jak również modeli wirtualnych w animacjach i grach. Osobnym zastosowaniem jest wspoma-

ganie procesów produkcyjnych CAD/CAM/CAE i w tym obszarze znajduje ono zastosowanie w projektowaniu architektonicznym i konstrukcyjnym. Dzięki rozszerzeniom dostępnym na stronie producenta system ten może być wykorzystywany w wielu dziedzinach i branżach, nie tylko przemysłowych.

Naturalną cechą środowiska Rhinoceros-Grasshopper jest parametryzacja projektowanego procesu, wykorzystywana właśnie w czasie projektowania inżynierskiego.

Rhinoceros jest używany zarówno przez profesjonalistów działających w branżach inżynieryjnych, jak i przez studentów, choćby architektury, podczas ich nauki i przygotowywania koncepcji oraz kształtowania form projektowanych obiektów. Osobnym zastosowaniem środowiska Rhino jest samo ideowe projektowanie parametryczne, zgodnie z koncepcją i definicją opisanymi wcześniej.

Istotnym elementem systemu jest technologia NURBS, która pozwala na tworzenie obiektów o praktycznie dowolnych kształtach i krzywiznach. Oparto ją na silniku AGLib NURBS, analogicznie jak aplikację Alias dla platformy Silicon Graphics.

Producent podaje następujące cechy środowiska Rhinoceros-Grasshopper [74]:

- nieograniczoność w zakresie tworzenia dowolnych kształtów;
- dokładność potrzebna do projektowania, prototypowania, analizowania i produkcji;
- zgodność ze wszystkimi innymi programami do projektowania, kreślenia, analiz, renderowania, animacji i ilustracji;
- odczyt i naprawa siatek, obsługa plików formatu IGES;
- dostępność i łatwość obsługi;
- duża szybkość działania.

Tak jak podano powyżej, środowisko Grasshopper cechuje wysoka kompatybilność z wieloma formatami stosowanymi w projektowaniu i modelowaniu 3D. Obsługuje ono ponad 30 różnych rodzajów formatów, takich jak DWG, DXF, OBJ, RIB, VRML, BMP, TGA, JPG, CSV i inne. To o tyle istotne z punktu widzenia inżyniera projektanta, że zapewnia przepływ informacji i danych pomiędzy najczęściej stosowanymi w tej branży programami.

Ważny jest również sam interfejs graficzny, czyli edytor do tworzenia grafiki trójwymiarowej. Pozwala on tworzyć i w łatwy sposób modyfikować podstawowe bryły składowe, np.: sześciany, walce, stożki czy elipsoidy, oraz elementy powierzchniowe, takie jak siatki czy powierzchnie swobodne.

Interesującą i wydajną opcją jest zastosowanie w środowisku Rhino technologii SPLOP i UDT, czyli *Universal Deformation Technology*. Pozwala ona na nakładanie jednego modelu na drugi w trójwymiarze. Zachowane przy tym zostają forma i proporcje poszczególnych modeli, które są odpowiednio do siebie dostosowywane, czyli sklonowane. Utrzymana przy tym jest możliwość dowolnej modyfikacji, deformacji elementów składowych czy obiektów w 3D.

Grasshopper oprócz szerokiego i ugruntowanego zastosowania w projektowaniu parametrycznym [75, 76], gdzie jest niekwestionowanym liderem w branży archi-

tektonicznej, coraz częściej zaczyna stanowić silnik projektowania generatywnego [77], które w tym momencie wydaje się być kierunkiem przyszłości.

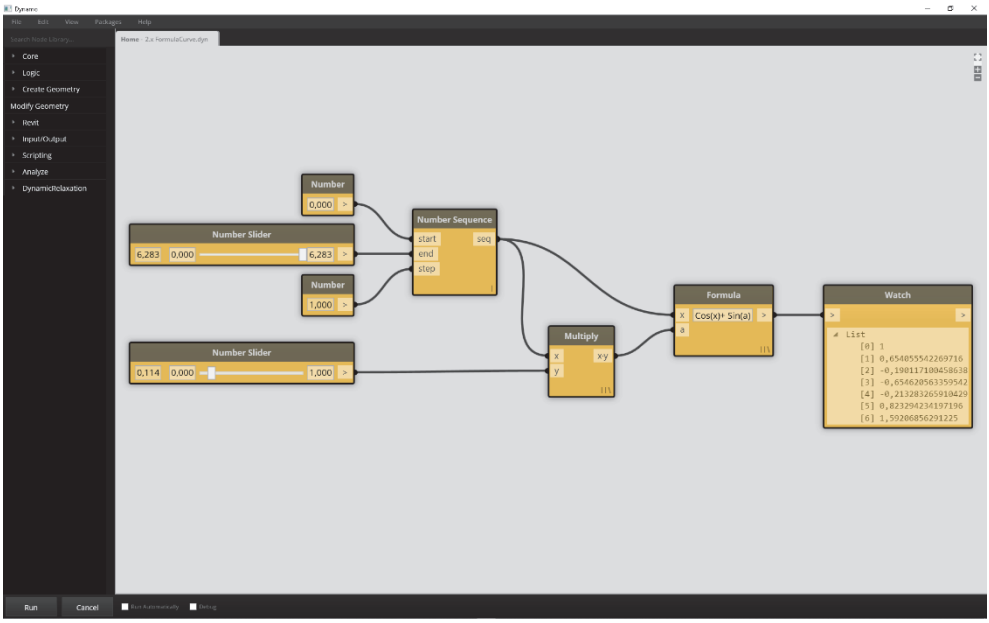
Podsumowując, należy stwierdzić, że Rhinoceros-Grasshopper to obecnie jedno z podstawowych środowisk programistyczno-projektowych stosowanych w projektowaniu inżynierskim, obejmującym przede wszystkim branże architektoniczną i konstrukcyjną, z przewagą tej pierwszej. Jak efektywne może być modelowanie parametryczne w środowisku Rhino w połączeniu z aktualnymi możliwościami renderowania, można przekonać się, podziwiając wizualizację stadionu pokazaną na rysunku 2.2.

Rys. 2.2. Efekt modelowania w środowisku Rhino [78]

Wydaje się, że z uwagi na funkcjonalność, ugruntowanie na rynku, zastosowania praktyczne w ramach wielu zrealizowanych projektów i obiektów Rhinoceros-Grasshopper to jeden z najpoważniejszych graczy w swojej kategorii. Jego pozycja wydaje się być niezagrożona w perspektywie kilku czy nawet kilkunastu lat. To system dopracowany, który można polecić zarówno zaawansowanym inżynierom projektantom, jak i dopiero co zaczynającym swoją karierę w branży.

2.2.2. Dynamo

Drugim środowiskiem, które jak do tej pory w najszerszym stopniu zadomowiło się w branży projektowej, jest Dynamo. Powstało ono jako samodzielna aplikacja oparta na programowaniu wizualnym, z zainstalowanym silnikiem, przez co nie wymaga dołączania go z zewnątrz (rys. 2.3). System Dynamo bazuje na języku DesignScript, które jest silnikiem budowania skryptów wizualnych.



Rys. 2.3. Dynamo w wersji 0.61

Z uwagi na rozwój tego programu został on dość dawno zaimplementowany i zintegrowany z innymi środowiskami. Dzięki temu istnieje możliwość wykorzystania go w różnych obszarach i etapach projektowania. Dynamo znajduje zastosowanie w obliczeniach, kształtowaniu geometrii i formy obiektów, zarządzaniu informacją BIM oraz tworzeniu dokumentacji technicznej, w tym do detalowania elementów konstrukcyjnych na potrzeby projektów wykonawczo-warsztatowych. Możliwości te są oczywiście o wiele szersze i wciąż dynamicznie się rozszerzają. Integracja Dynamo dotyczy przede wszystkim programu Autodesk Revit, a więc jednego z największych i najpopularniejszych środowisk BIM na świecie. Pierwsza możliwość korzystania z Dynamo w Revicie miała miejsce w wersji 2013 [79], a więc dobre już kilka lat temu. Inne programy, które włączyły do współpracy opcję programowania wizualnego w wersji Dynamo, to Autodesk Advance Steel, czyli specjalistyczne środowisko specjalnie przeznaczone detalowaniu elementów konstrukcji stalowych, czy ostatnio Autodesk Robot Structural Analysis, gdzie do wersji 2022 dodano wtyczkę Dynamo jako składnik programu.

System kodowania wizualnego Dynamo oparto na dwóch anatomicznych elementach, tj. węzłach (*nodes*) i przewodach (*wires*), podobnie jak i inne tego typu środowiska i języki klasy VPL. Zostanie to szczegółowo omówione w kolejnym rozdziale, gdzie opisano ideę programowania wizualnego w środowisku Dynamo, jak również jego interfejs, dostępne biblioteki węzłów i pozostałe składniki.

Środowisko Dynamo, tak jak i inne jemu podobne, umożliwia tworzenie prostych, wręcz elementarnych skryptów – takich jak przykładowo narzędzia do prze-

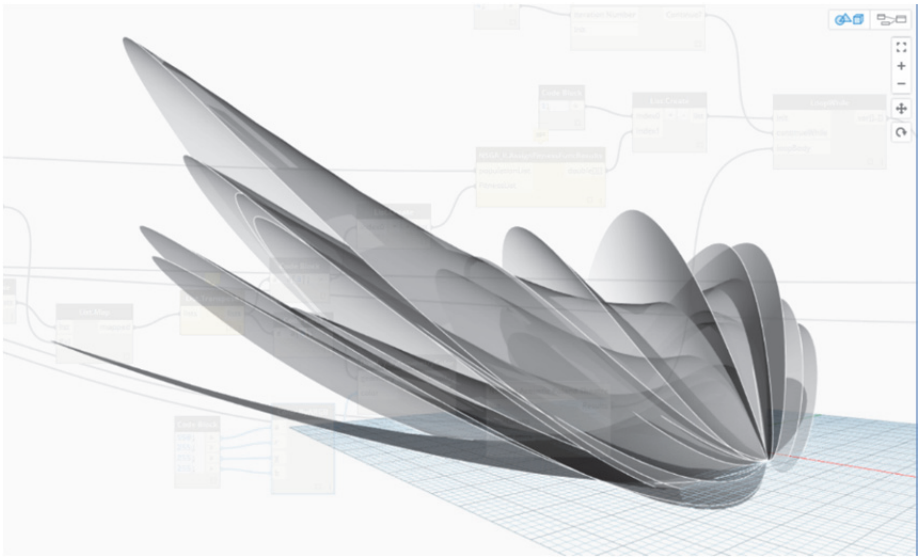
prowadzania operacji matematycznych (por. rys. 1.9) czy operacji na listach lub ciągach znaków. Główną ideą tego systemu jest możliwość budowania bardziej zaawansowanych kodów. Jednym z głównych celów w tym zakresie jest naturalna opcja i koncepcja parametrycznego i algorytmicznego opisu modeli tworzonych obiektów. Dotyczy to przede wszystkim struktur 2D i 3D, za pomocą których modelowane są obiekty budowlane oraz kształtowane formy architektoniczne i konstrukcje.

Spośród wachlarza funkcjonalności środowiska Dynamo najciekawszą i najefektywniejszą opcją wydaje się możliwość sparametryzowanego, a tym samym elastycznego modelowania i kształtowania geometrii projektowanych struktur. Funkcjonalność ta jest możliwa właśnie dzięki parametrycznemu opisowi, który pozwala na łatwą edycję i modyfikację geometrii poprzez zmianę parametrów sterujących. W tym zakresie Dynamo bardzo dobrze sprawdza się jako system modelowania i projektowania struktur niesymetrycznych, o geometrii dowolnej 3D, w tym rozwijanej od wielu już lat formie organicznej różnej postaci. Matematyczny opis elementów składowych determinujących tego typu geometrie w postaci krzywych i powierzchni typu NURBS pozwala poprzez zmianę punktów sterujących na płynną i elastyczną optymalizację kształtu i formy projektowanych obiektów. To bardzo wydajne narzędzie podczas projektowania struktur architektonicznych i konstrukcyjnych, zarówno na etapie koncepcyjnym, jak i na przykład podczas detalowania ustrojów nośnych.

Inne zastosowania systemu Dynamo to sterowanie i edycja modeli BIM, np. w systemie Revit. W tym zakresie skrypt wizualny pozwala na porządkowanie informacji w modelu, edycję elementów składowych, edycję rodzin Revit, zarządzenie informacją BIM w projekcie w bardzo szerokim zakresie. Efektywną opcją jest także praca z „chmurą" punktów, za pomocą której można modelować choćby obiekty inżynierskie czy elementy liniowe egzystujące w branży infrastrukturalnej (drogi i mosty).

Osobną funkcjonalnością, która zasługuje na uwagę, jest projektowanie generatywne Dynamo zintegrowane ze środowiskiem BIM. Po zainstalowaniu odpowiedniego dodatku pod nazwą Generative Design system pozwala na wariantową optymalizację procesu projektowania i zarządzania modelem w systemie Revit, z którym w tym zakresie Dynamo jest zintegrowane. Programowanie generatywne jest w tym zestawie (Dynamo + Revit) rozwijane od jakiegoś czasu, a patrząc na ogólne trendy, jakie mają miejsce w projektowaniu komputerowym, kwestią czasu jest takie upowszechnienie tego podejścia, aby na każdym etapie i w najważniejszych aplikacjach znajdowały się opcje pozwalające na tego typu optymalizację.

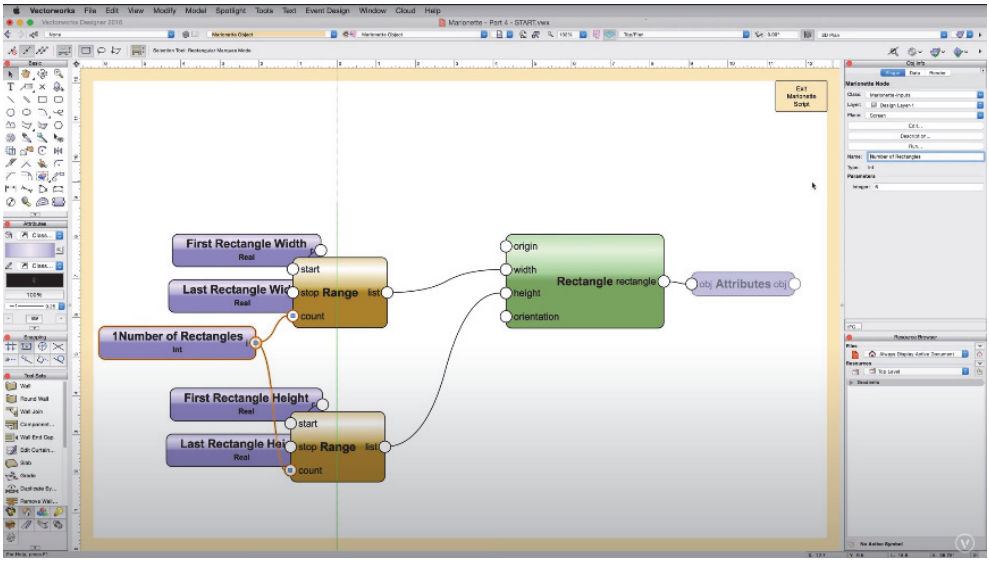
Środowisko Dynamo, jak i jemu podobne, daje projektantowi nieograniczone w zasadzie możliwości wykorzystania w optymalizacji różnorakich obszarów. Przykładem może być choćby poszukiwanie i znalezienie optymalnej geometrii zadaszenia letniego teatru w Szczecinie (rys. 2.4).



Rys. 2.4. Wariantowość geometrii zadaszenia letniego teatru w Szczecinie [80]

2.2.3. Vectorworks-Marionette

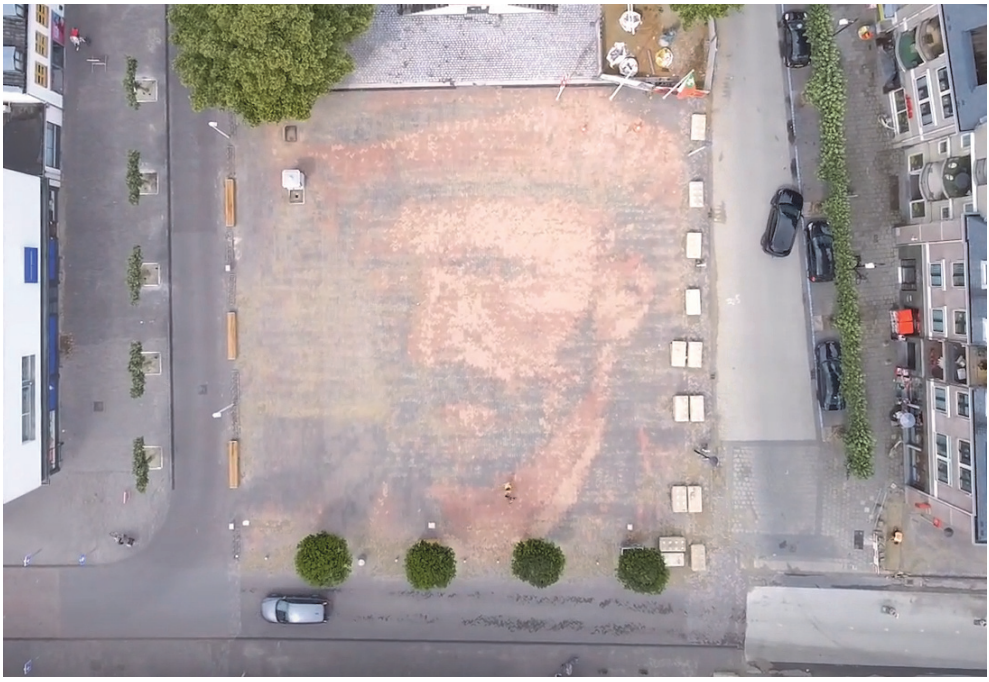
Kolejnym systemem opartym na programowaniu wizualnym jest Vectorworks-Marionette, gdzie Marionette to język VPL, a Vectorworks to środowisko projektowe CAD/BIM produkowane przez Nemetschek North America.



Rys. 2.5. Środowisko Vectorworks-Marionette [81]

Producent określa ten system jako pierwsze wieloplatformowe środowisko programowania wizualnego dostępne w oprogramowaniu do tworzenia BIM dla branży AEC, czyli *Architecture, Engineering and Construction*, rozrywki i krajobrazu [82]. Tak jak poprzednio opisane środowiska jest ono oparte na sieci (skrypcie) będącej zbiorem węzłów, z których każdy reprezentuje funkcję lub mapowanie z wejść na wyjścia (rys. 2.5).

System Vectorworks-Marionette ma szerokie zastosowanie, które producent podzielił w trzech wymienionych wyżej grupach (*Buildings*, *Landscapes*, *Entertainment*), jednakże przykłady zastosowań wykraczają daleko poza te kategorie. Przykładem może być Zundert w Holandii, miasto rodzinne Vincenta van Gogha, gdzie na placu przed jego domem wykonano portret artysty z kostki brukowej (rys. 2.6). Opracowany algorytm pozwolił na rozplanowanie rozmieszczenia poszczególnych kostek tak, aby osiągnąć efekt w postaci jego autoportretu.



Rys. 2.6. Efekt zastosowania środowiska Marionette do zaprojektowania portretu Vincenta van Gogha ułożonego z kostki brukowej na placu w Zundert w Holandii [83]

Samo środowisko i język jest przyjazny dla użytkownika i przy niewielkiej wiedzy z zakresu programowania inżynier projektant ma możliwość tworzenia różnych, niekoniecznie standardowych algorytmów wykorzystywanych podczas samego procesu projektowania oraz do zarządzania informacjami. To istotne, tak jak to już opisano, gdyż zachęca do korzystania z narzędzi opartych na językach VPL, przyspieszając pracę projektanta i automatyzując sam proces projektowy.

2.2.4. GenerativeComponents

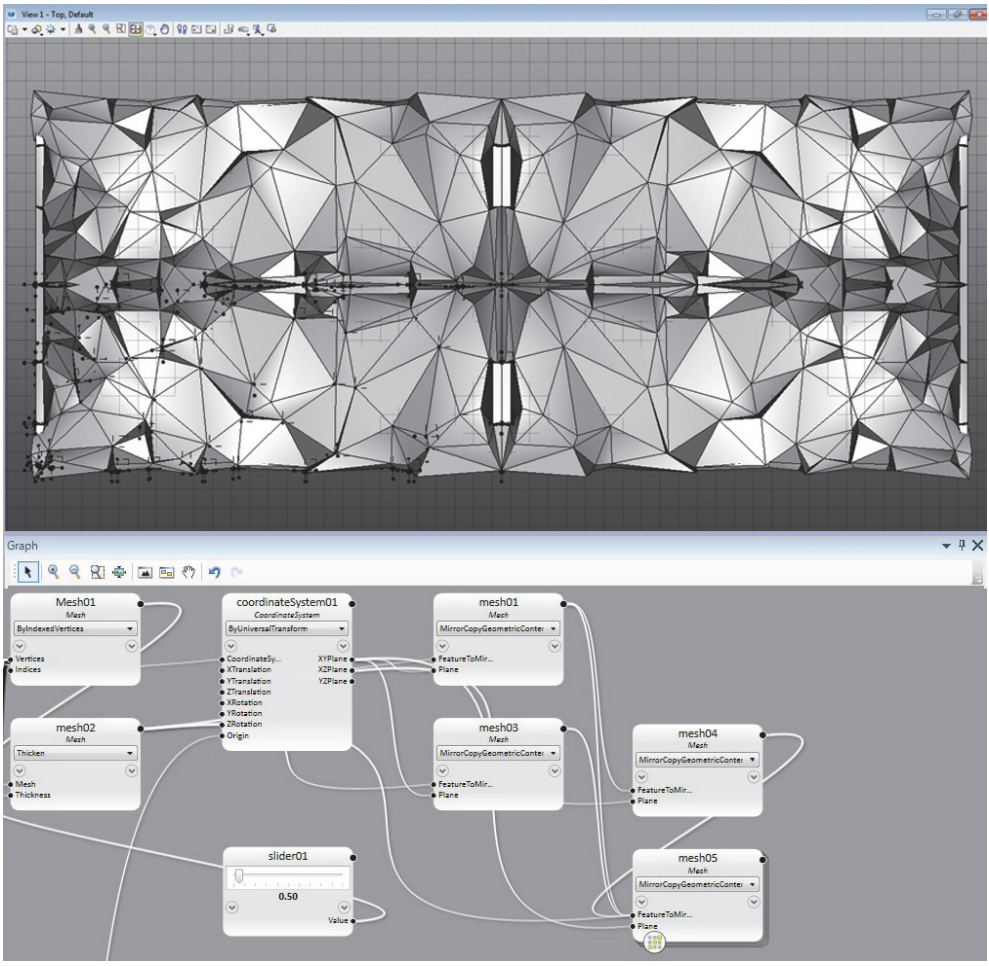
Nieco innym systemem jest przeznaczony głównie zastosowaniom optymalizacyjnym GenerativeComponents. To produkt topowego producenta oprogramowania CAD/BIM i nie tylko, a mianowicie firmy Bentley Systems. Jego największą siłą jest możliwość zastosowania algorytmów parametrycznych służących projektowaniu generatywnemu. Ten asocjacyjny i parametryczny system modelowania jest używany przede wszystkim przez architektów i inżynierów do automatyzacji procesów projektowych głównie w oparciu o iteracje projektowe, które dzięki zastosowaniu inteligentnych algorytmów ulegają znaczącemu przyspieszaniu. Pozwala to na efektywne eksplorowanie alternatywnych form budynku, które są generowane automatycznie w oparciu o wybrany algorytm oraz ustawione parametry optymalizacji. Nie ma zatem konieczności ręcznego budowania modelu projektu szczegółowego dla każdego przyjętego założenia. System GenerativeComponents pozwala również na zwiększenie efektywności w zarządzaniu konwencjonalnym projektowaniem i dokumentacją.

Projektowanie w systemie GenerativeComponents obejmuje zarówno komponenty projektu, jak i abstrakcyjne relacje między nimi, dzięki czemu projektanci mogą wykraczać poza geometrię zdeterminowaną i zorientowaną przestrzennie. Opis modelu jawnie geometrycznego jest ujęty w postaci logicznego algorytmu.

System GenerativeComponents oparto na alternatywnym, hybrydowym podejściu do projektowania, gdzie proces modelowania jest możliwy w oparciu o kilka metod:

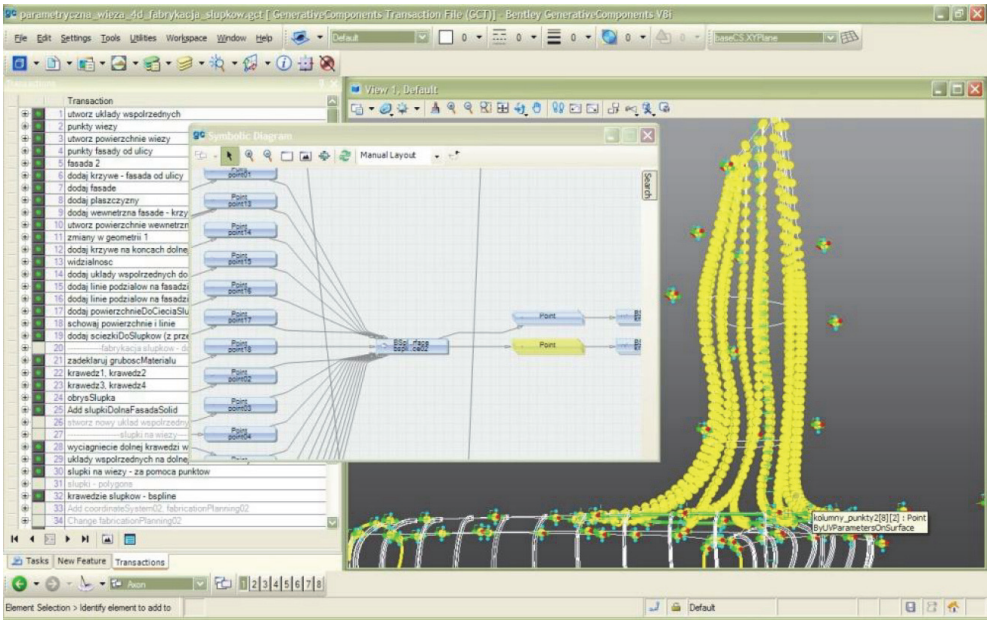
- graficznie interaktywne, praktyczne modelowanie z bezpośrednią manipulacją (*graphic-interactive*, *hands-on modeling with direct manipulation*);
- programowanie graficzne lub skryptowanie w formie diagramu (*graphic programming or scripting in diagrammatic form*);
- zwięzłe wyrażenia, zależności warunkowe i skrypty obiektowe (*concise expressions*, *conditional relationships*, *and object-oriented scripting*);
- rozbudowa systemu o .Net programowanie (*extension of the system using .Net programming*).

System GenerativeComponents pokazano na rysunku 2.7, a anatomia tego środowiska jest analogiczna do prezentowanych w poprzednich podrozdziałach.



Rys. 2.7. Środowisko programistyczne systemu GenerativeComponents [84]

Innym zastosowaniem, a raczej rozszerzeniem systemu GenerativeComponents jest rozwój narzędzi projektowych. Dotyczy to przeznaczonych dla poszczególnych zastosowań, branż, przypadków szczególnych algorytmów. Wpisuje się to w dyskutowaną wcześniej specjalizację poszczególnych narzędzi opartych na programowaniu wizualnym, takich jak specyficzne węzły, pakiety węzłów o specjalnym przeznaczeniu czy wreszcie bardziej kompletne i zaawansowane rozszerzenia (rys. 2.8).



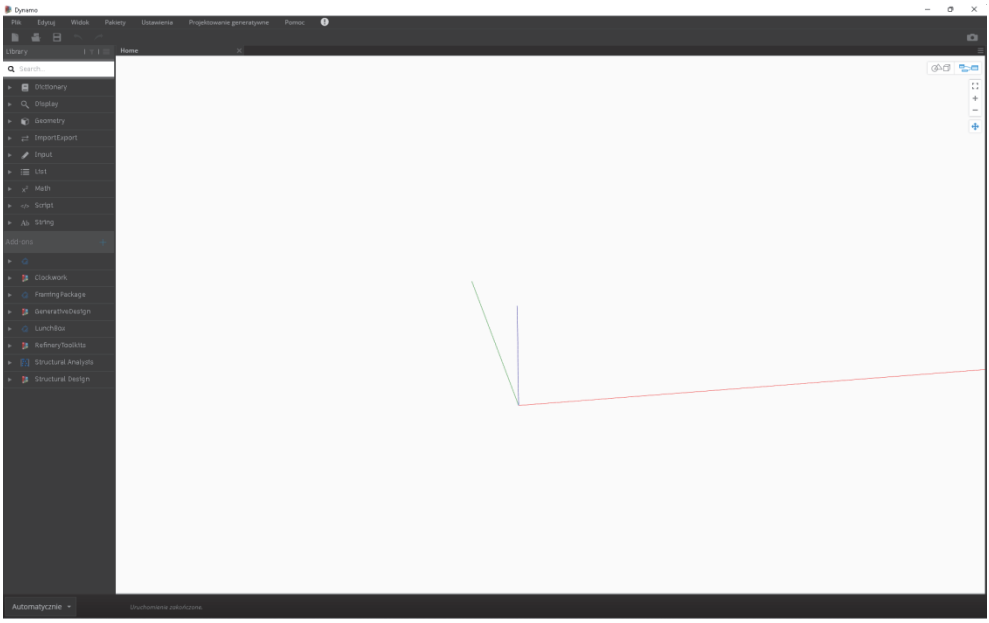
Rys. 2.8. Przykład zastosowania systemu GenerativeComponents w projektowaniu inżynierskim [85]

3 ŚRODOWISKO PRACY SYSTEMU DYNAMO

Z omówionych w poprzednim rozdziale programów komputerowego wspomagania projektowania, które wykorzystują programowanie wizualne, najczęściej znajdującym zastosowanie w projektowaniu konstrukcyjnym jest środowisko Dynamo. Niejako na jego przykładzie omówione zostanie podejście do projektowania inżynierskiego oparte na kodowaniu VPL oraz poruszanie się w tym środowisku. Z praktycznego, ale i konceptualnego punktu widzenia w dalszej części zaprezentowano przykłady ilustrujące tworzenie skryptów umożliwiających modelowanie podstawowych, jak i bardziej skomplikowanych struktur konstrukcyjnych. Dzięki temu użytkownik wchodzący w temat programowania wizualnego ma możliwość jak najlepszego zapoznania się z tym podejściem i technologią.

3.1. INTERFEJS

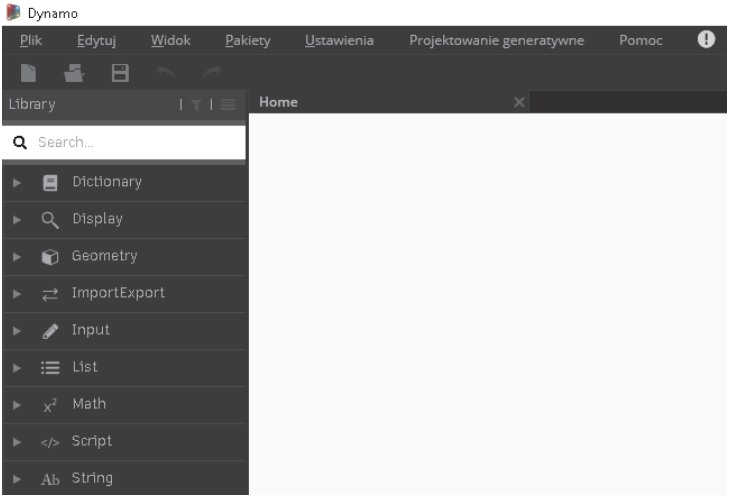
Architektura środowiska programistycznego Dynamo jest typowa i podobna do środowisk programowania wizualnego stworzonych i używanych obecnie na świecie. Interfejs programu Dynamo Sandbox w wersji 2.10 pokazano na rysunku 3.1. Układ jest analogiczny do innych wersji aplikacji dostępnych jako samodzielny program Dynamo Studio czy Dynamo Revit, będący zintegrowanym elementem środowiska Revit.



Rys. 3.1. Interfejs środowiska programistycznego Dynamo Sandbox 2.10

Jak widać, główną część interfejsu zajmuje obszar roboczy przeznaczony do budowania skryptu i wizualizacji efektów jego działania. Tym samym praca w środowisku Dynamo odbywa się w dwóch przestrzeniach: skryptu i wizualizacji jego wyniku. Użytkownik przełącza się pomiędzy nimi za pomocą ikon umieszczonych w górnym prawym narożniku przestrzeni roboczej.

Górny pasek zawiera menu rozwijane (rys. 3.2), gdzie umieszczono podstawowe funkcje, służące do zarządzania plikami i przeprowadzania innych operacji.



Rys. 3.2. Menu rozwijane

Użytkownik ma także możliwość uruchomienia bardziej specyficznych funkcji. W kolejności od lewej umieszczono w nim opcje *Plik*, *Edycja*, *Widok*, *Pakiety*, *Ustawienia*, *Pomoc* i *Powiadomienia*.

Ikony umieszczone na pasku narzędzi pod menu rozwijanym, po lewej stronie, umożliwiają szybki dostęp do plików oraz uruchomienie poleceń *Cofnij* [Ctrl+Z] i *Ponów* [Ctrl+Y]. Po prawej, przeciwległej stronie umieszczona jest ikona „spustu migawki" *Eksportuj obszar roboczy/podgląd tła jako obraz*, za pomocą której użytkownik ma możliwość zapisu właśnie obszaru roboczego lub podglądu tła jako grafiki w formacie *PNG. Funkcja ta jest przydatna w sytuacji, gdy trzeba zapisać np. cały stworzony skrypt, a standardowy zrzut ekranu nie gwarantuje odpowiedniej czytelności takiego obrazu.

Z programistycznego punktu widzenia podstawowym elementem systemu jest biblioteka procedur, która znajduje się po lewej stronie pod paskiem narzędzi. Zawarto w niej wszystkie dostępne w środowisku procedury programistyczne, które można wywołać, bezpośrednio klikając na poszczególne kategorie lub wpisując w pole wyszukiwania słowa kluczowe. Zaletą tej drugiej metody jest szybkie wczytanie wszystkich pasujących komend, co znacznie ułatwia ich odszukanie i wywołanie. W bibliotece zawarto również procedury niestandardowe oraz pakiety

będące zbiorem procedur instalowanych ze źródeł zewnętrznych. Organizacja procedur w bibliotece jest hierarchiczna, zgodnie z podziałem na poszczególne typy i grupy. Podział ten określa dane kategorie, podkategorie itd., zgodnie z ogólnym podziałem dotyczącym danych przetwarzanych przez węzły, tj. tworzenia danych, wykonywania na nich operacji i wysyłania zapytań dotyczących danych.

3.2. BUDOWA SKRYPTU DYNAMO, JĘZYK DESIGNSCRIPT

System Dynamo, we wszystkich wydaniach i wersjach, oparto na języku programowania wizualnego o nazwie DesignScript.

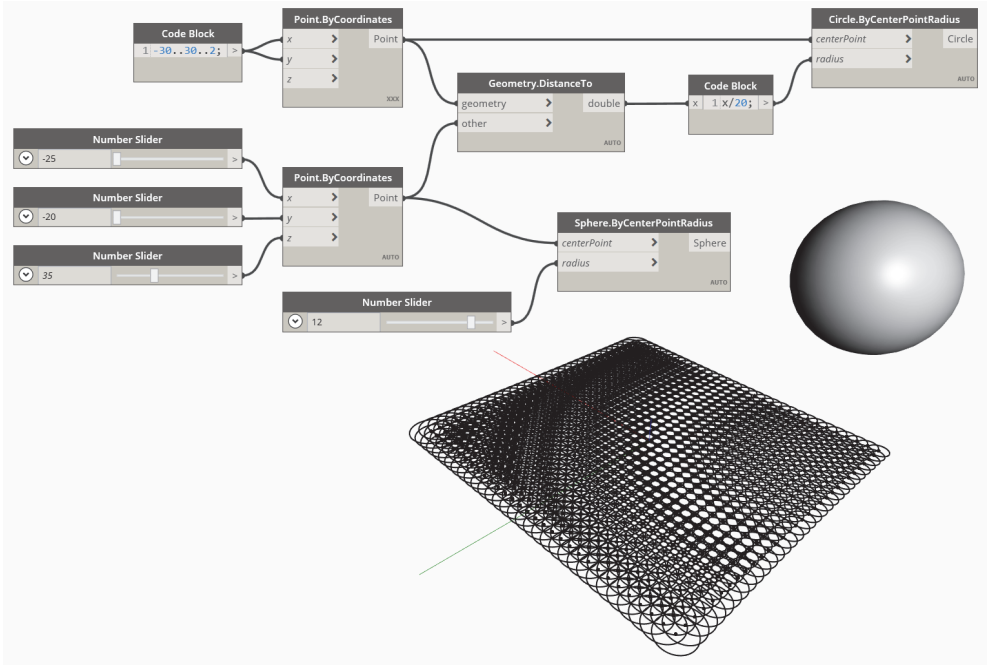
Dynamo to *de facto* zintegrowane środowisko programistyczne, czyli IDE (ang. *Integrated Development Environment*), przewidziane do tworzenia, modyfikacji i testowania skryptów opartych na programowaniu wizualnym.

Jak już wspomniano, interfejs użytkownika obejmuje dwie przestrzenie robocze, przestrzeń skryptu i podglądu geometrii tworzonego modelu.

Anatomia skryptu oparta została na dwóch podstawowych elementach:

- węzłach (*nodes*), w których zakodowane są procedury przetwarzania danych;
- przewodach (*wires*), będących łączami odpowiedzialnymi za przesyłanie danych pomiędzy węzłami.

Skrypt wizualny Dynamo stanowi więc szereg węzłów połączonych ze sobą przewodami, tworząc mniej lub bardziej skomplikowany organizm (rys. 3.3).



Rys. 3.3. Struktura skryptu wizualnego Dynamo

Przepływ danych jest kontrolowany na bieżąco, zgodnie z regułami określonymi składnią języka DesignScript. Obliczenia są przeprowadzane w węzłach, do których są dostarczane dane z innych węzłów lub źródeł zewnętrznych.

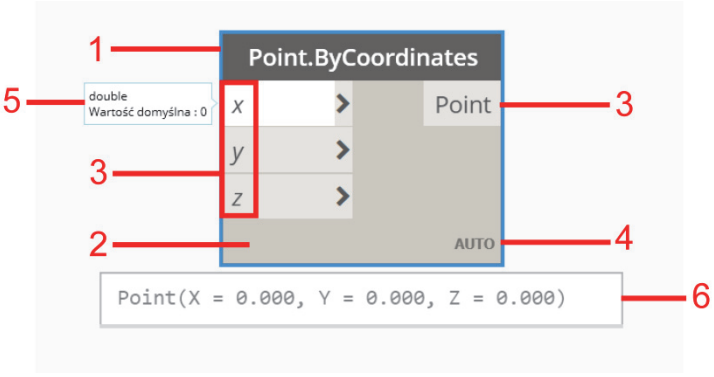
3.2.1. Węzły

Węzeł to podstawowy element skryptu wizualnego Dynamo. Jest w nim zakodowana procedura odpowiedzialna za przetwarzanie danych. Jest to odpowiednik składni tradycyjnego skryptu tekstowego w postaci komendy, procedury, za pomocą której możliwe jest przeprowadzanie różnorakich operacji na danych. Operacje wykonywane przez węzły na danych cechują się różnym poziomem skomplikowania. Może to być zarówno przechowywanie danych wprowadzonych przez użytkownika bezpośrednio do węzła, np. definicja liczby czy współrzędnych punktu, jak i skomplikowane operacje matematyczne na listach, tablicach czy przekształcanie geometrii.

3.2.1.1. Budowa węzła

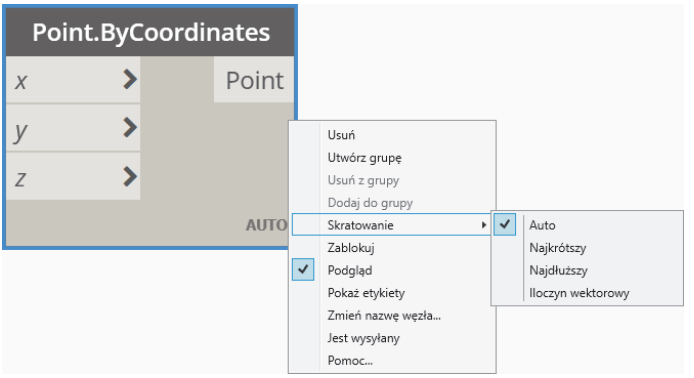
Węzeł skryptu wizualnego Dynamo ma postać bloku graficznego, który można dowolnie umieszczać i przesuwać w przestrzeni roboczej programu. Standardowy węzeł składa się z sześciu elementów, które zaznaczono na rysunku 3.4. Są to:

1. *Nazwa* – oznaczenie węzła zgodne z konwencją nazewnictwa określonego przez daną kategorię.
2. *Część główna* – treść węzła.
3. *Porty* – wtyczki wejścia i wyjścia, służące do podłączania przewodów odpowiadających za przepływ danych.
4. *Skratowanie* – opcja zarządzania danymi określonymi w liście wejściowej.
5. *Wartość domyślna* – opcja dostępna dla portów wejściowych niektórych węzłów.
6. *Podgląd* – wizualizacja wyniku działania węzła.



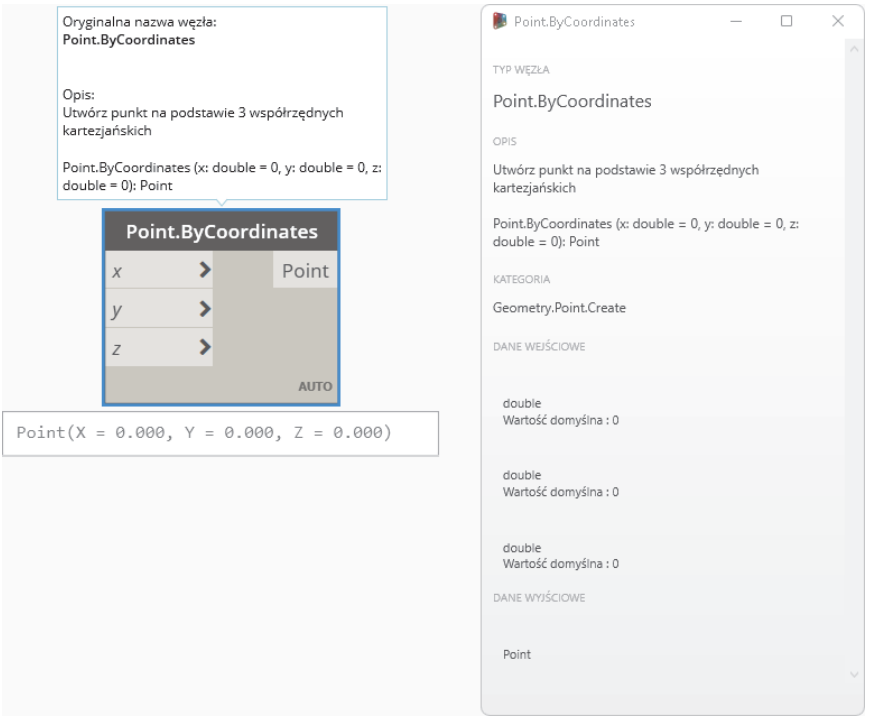
Rys. 3.4. Budowa węzła

Zarządzanie opcjami węzła jest możliwe za pomocą menu podręcznego, uruchamianego prawym klawiszem myszy. Użytkownik ma do dyspozycji kilka parametrów w tym zakresie, w tym zarządzanie *Skratowaniem* danych (rys. 3.5).



Rys. 3.5. Menu węzła

Podstawowe informacje na temat danego węzła są dostępne po najechaniu myszką na pole (1) *Nazwa*. Użytkownik ma również możliwość uruchomienia szybkiej pomocy, aktywując ją z menu podręcznego (rys. 3.6).



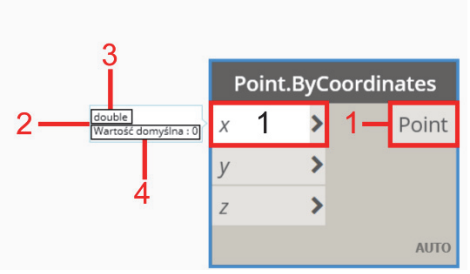
Rys. 3.6. Informacje o węźle

3.2.1.2. Porty

Dane wprowadzane są do węzła poprzez tzw. porty wejściowe (*input*), umieszczone po jego lewej stronie. Są to niejako gniazda, do których podłączane są wtyczki przewodów służących do transmisji danych. Po wprowadzeniu do węzła są one poddane przetwarzaniu. Po zakończeniu tego procesu dane wynikowe są wysyłane dalej za pośrednictwem portów wyjściowych (*output*), umiejscowionych po prawej stronie węzłów.

Porty składają się z czterech podstawowych elementów takich jak (rys. 3.7):

1. *Etykieta portu.*
2. *Etykieta narzędzia.*
3. *Typ danych.*
4. *Wartość domyślna.*



Rys. 3.7. Elementy portów

W kwestii obsługi portów i w ogóle przetwarzania danych w węźle podstawową sprawą jest wprowadzenie do węzła odpowiedniego typu danych. Jest to niejako oczekiwanie portów wejściowych, aby otrzymać dane odpowiedniego rodzaju. W przypadku wprowadzenia danych nieodpowiedniego typu skutkuje to błędem, który jest od razu sygnalizowany.

W zależności od rodzaju i funkcjonalności węzła na wejściu podawane są domyślne wartości wprowadzanych do portów danych. W przypadku węzła pokazanego na rysunku 3.7, definiującego punkt na podstawie jego współrzędnych x, y, z, przyjmują one wartości 0, a w wyniku działania utworzony jest punkt o współrzędnych (0, 0, 0). Można to sprawdzić na podglądzie działania węzła (por. rys. 3.4).

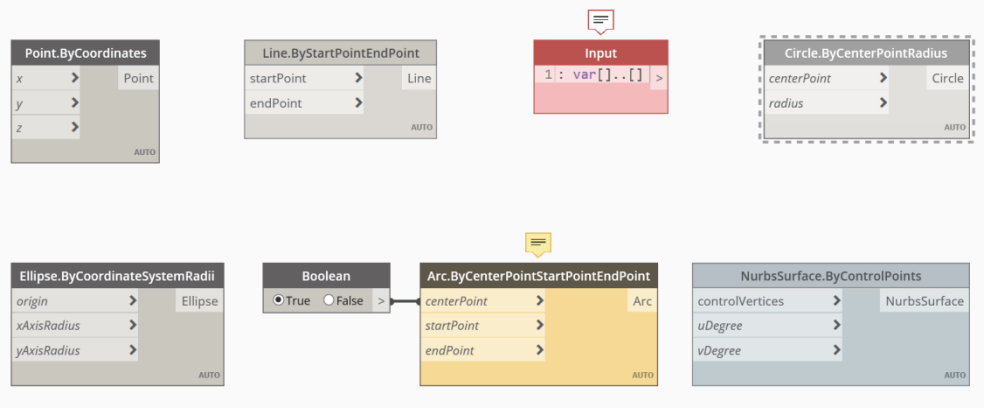
3.2.1.3. Stany pracy węzłów

Aktualny status danego węzła określany jest jego tzw. stanem (*state*). Odzwierciedla on etap lub sytuację dotyczącą przetwarzania danych w danych miejscu skryptu. W zależności od aktualnej fazy, zaistnienia błędów czy innych czynników determinujących proces przetwarzania danych zmianie ulega stan danego węzła. Jest to sygnalizowane zmianą jego wyglądu, przede wszystkim koloru, oraz ewentualnym pojawieniem się określonych monitów.

W programie Dynamo istnieje siedem podstawowych stanów określających status węzłów:

1. *Aktywne* – to węzły poprawnie połączone, przepływ danych jest poprawny, oznaczeniem jest ciemnoszare tło nazw węzłów.
2. *Nieaktywne* – węzły, które nie zostały połączone, brak przepływu danych, oznaczeniem jest jasnoszare tło nazw węzłów.
3. *Stan błędu* – węzeł w stanie błędu, oznaczeniem jest czerwony kolor całego węzła.
4. *Zablokowanie* – zawieszone wykonanie węzła, oznaczeniem jest przezroczystość, wyszarzenie węzła oraz przerywana obwódka.
5. *Wybrane* – węzeł poddany selekcji, oznaczeniem jest błękitne obramowanie węzła.
6. *Ostrzeżenie* – niepoprawnie określony typ węzła (typ danych), oznaczeniem jest kolor żółty węzła.
7. *Podgląd w tle* – wyłączony podgląd geometrii, oznaczeniem jest kolor ciemnoszary.

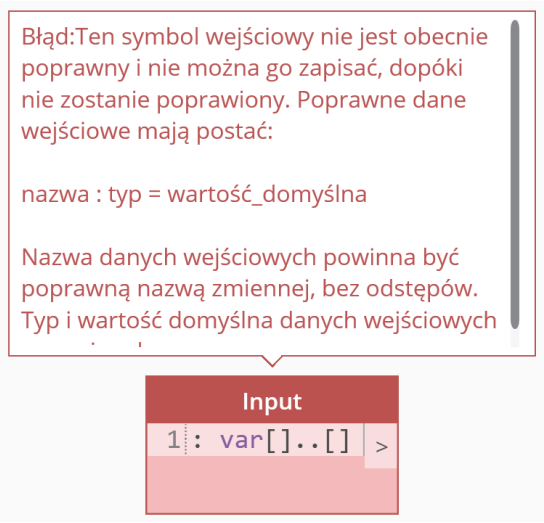
Przykłady wyżej wymienionych stanów dla różnych węzłów pokazano rysunku 3.8.



Rys. 3.8. Stany węzłów

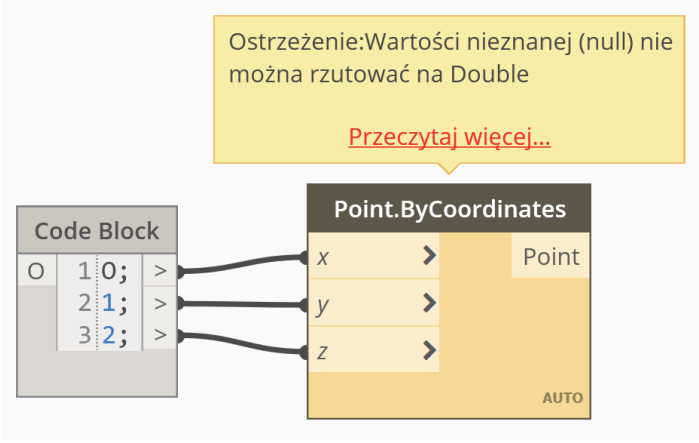
W sytuacji możliwości wystąpienia błędów węzeł, którego to dotyczy, zmienia swój status na stan *błędu* lub *ostrzeżenia*, co powoduje również wyświetlenie krótkiej informacji na temat przyczyn takiej sytuacji.

Na rysunku 3.9 pokazano przykład węzła typu *Input*, gdzie błędnie zdefiniowano dane wejściowe. System podaje informację o przyczynie takiego, a nie innego stanu węzła (błąd) oraz stara się wskazać sposób rozwiązania problemu. W tym przypadku jest to kwestia poprawy danych wejściowych.



Rys. 3.9. Węzeł w stanie błędu

Przykładowy stan *ostrzeżenia* pokazano na rysunku 3.10. W tym przypadku niepoprawnie zdefiniowano współrzędną punktu *x* jako O, a nie 0.



Rys. 3.10. Węzeł w stanie ostrzeżenia

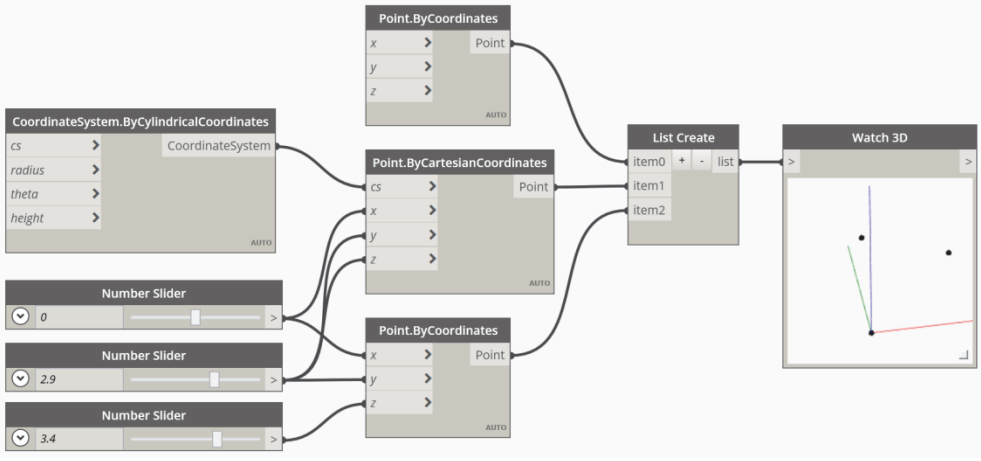
Z programistycznego punktu widzenia sygnalizowanie błędów i ostrzeżeń w czasie rzeczywistym, a przede wszystkim w sposób graficzny jest bardzo pomocne. Pozwala od razu zdiagnozować miejsce ewentualnego problemu w strukturze skryptu oraz zidentyfikować jego przyczyny. Pomocne są tutaj wskazówki określające przyczynę problemów, jak również wskazówki dotyczące możliwości ich korekty.

3.2.2. Przewody

Jak już wspomniano, węzły są łączone ze sobą za pomocą tzw. przewodów (*wires*). Są to elastyczne łączniki, stanowiące niejako kable, za pomocą których odbywa się przepływ danych pomiędzy połączonymi przewodami węzłami. Możliwe jest łączenie pojedynczych węzłów dzięki pojedynczym przewodom, jak i większej liczbie węzłów. Zależy to od możliwości logicznego połączenia węzłów i ograniczeń z tym związanych (np. ograniczeni logicznych) oraz koncepcji przepływu danych w skrypcie.

3.2.2.1. Przepływ programu wizualnego

Transmisja danych z jednego węzła do drugiego odbywa się za pomocą przewodów. Proces ten jest określany jako tzw. przepływ programu wizualnego, choć po angielsku termin ten brzmi jako *data flow*, a więc przepływ danych. Należy zaznaczyć, że przepływ danych jest uporządkowany, z uwagi na organizację i funkcję opisanych wyżej portów. Dane są wprowadzane do węzła za pomocą przewodu podłączonego do portu wejściowego (rys. 3.11). Dalsza transmisja danych (ich przepływ) może następować na skutek wyprowadzenia przewodem z portu wyjściowego danych wynikowych. Dane te najczęściej służą obróbce w kolejnych węzłach, prowadząc do fazy wynikowej całego algorytmu, czyli uzyskania pożądanego efektu finalnego.

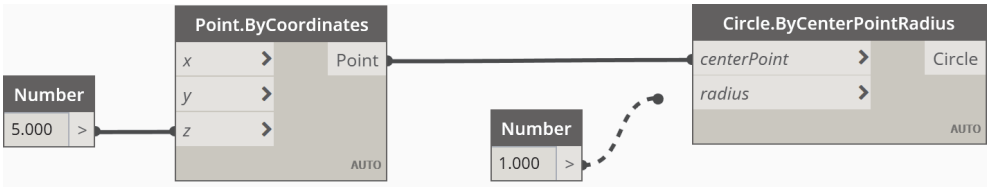


Rys. 3.11. Przepływ programu wizualnego

Należy również zaznaczyć, że z danego węzła można wyprowadzać dane niekoniecznie do jednego węzła, ale do wielu węzłów. Z uwagi na umiejscowienie portów wejściowych po lewej, a portów wyjściowych po prawej stronie węzłów funkcjonuje określony kierunek przepływu danych. Mówi się wtedy o przepływie od lewej do prawej strony.

3.2.2.2. Tworzenie i edytowanie przewodów

Sposób tworzenia i podłączania przewodów do węzłów w środowisku Dynamo jest bardzo prosty i intuicyjny. Wystarczy po prostu kliknąć lewym przyciskiem myszy odpowiednie porty dwóch węzłów, które chce się połączyć (rys. 3.12). Zgodnie z koncepcją transmisji danych będą one przepływać przez utworzony w ten sposób przewód z portu wyjściowego węzła do portu wejściowego połączonego węzła. Użytkownik ma możliwość przyjęcia dowolnego kierunku tworzenia przewodów, w zależności od umiejscowienia łączonych portów, ich sekwencji oraz umiejscowienia samych węzłów.



Rys. 3.12. Tworzenie i podłączanie przewodów

Użytkownik ma również możliwość zmiany połączeń pomiędzy węzłami. Jest to możliwe, dlatego że przewody są edytowalne. W tym zakresie do dyspozycji są dwie opcje:

- zmiana połączeń, czyli odłączenie przewodu od węzła i podłączenie go do innego węzła lub
- usunięcie przewodu.

3.3. BIBLIOTEKA WĘZŁÓW

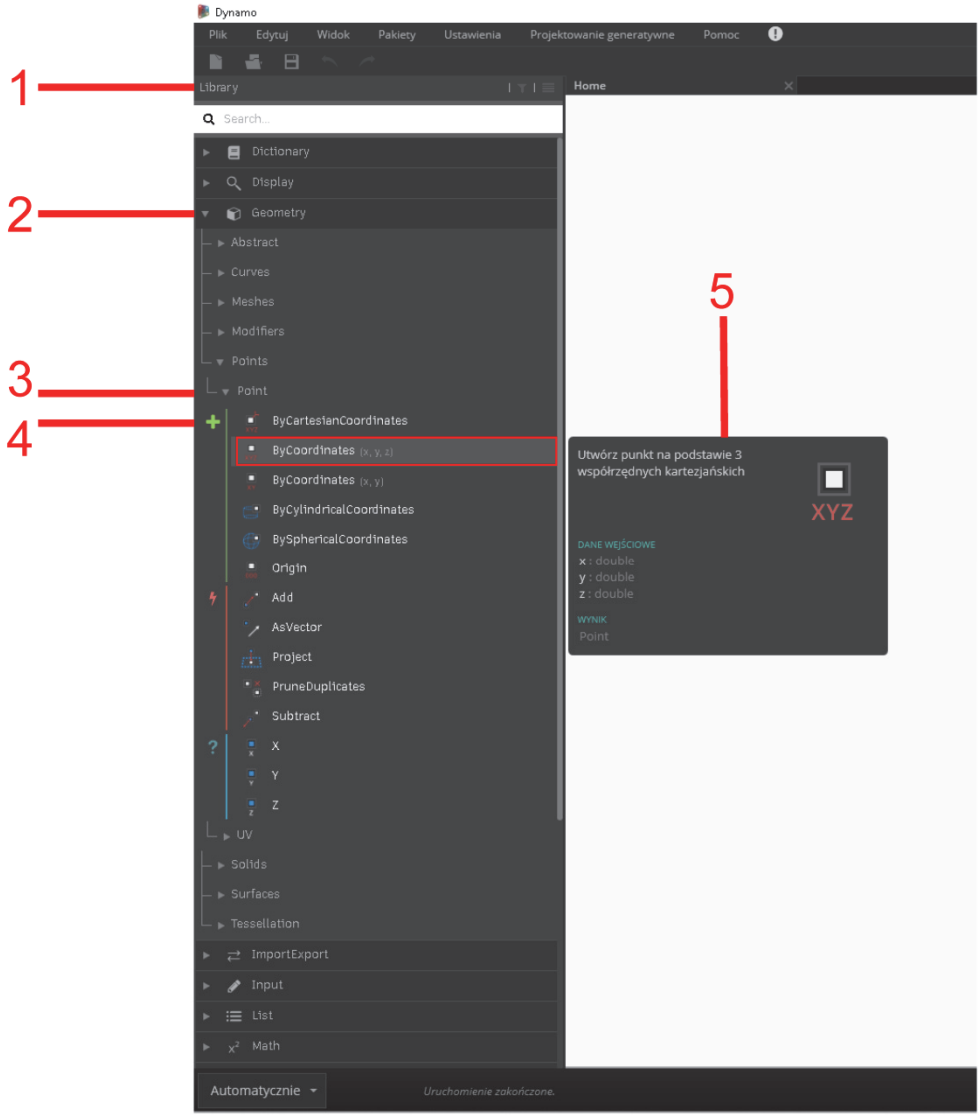
Opisane w poprzednich podrozdziałach węzły i łącza to tak naprawdę elementy składowe/narzędzia, za pomocą których dokonywane jest przetwarzanie i transmisja danych, umożliwiające osiągnięcie postawionego w programie celu, czyli uzyskanie pożądanego wyniku. Jak już wielokrotnie wspomniano, w węzłach zakodowano procedury w postaci komend, które tego przetwarzania danych dokonują. W Dynamo, jak i w każdym języku programowania komendy te są elementami jego składni, tyle że zwizualizowane. W tekstualnych językach programowania poszczególne komendy mają postać określonych słów czy ich skrótów, które przywołane powodują wykonanie odpowiednich operacji na danych, którymi mogą być liczby, elementy geometryczne, obiekty graficzne, ciągi alfanumeryczne i inne. Komendy zakodowane w węzłach zostały zorganizowane w tzw. bibliotece, o której już była wcześniej mowa przy omawianiu budowy środowiska Dynamo. Tak naprawdę biblioteka węzłów (komend) to podstawowy rezerwuar procedur, za pomocą których tworzony jest skrypt VPL w tym środowisku. Projektant programista w intuicyjny często sposób ma możliwość znalezienia i wybrania danej komendy (węzła) spośród wielu możliwych i zastosowania jej do przeprowadzenia żądanej operacji na danych.

3.3.1. Organizacja biblioteki

Biblioteka węzłów zorganizowana jest w programie Dynamo hierarchicznie. Układ poszczególnych poziomów jest następujący:

Biblioteka → Kategorie biblioteki → Podkategorie kategorii → Węzły

Pokazano to schematycznie na rysunku 3.13 na przykładzie węzłów zgrupowanych w kategorii *Point*.



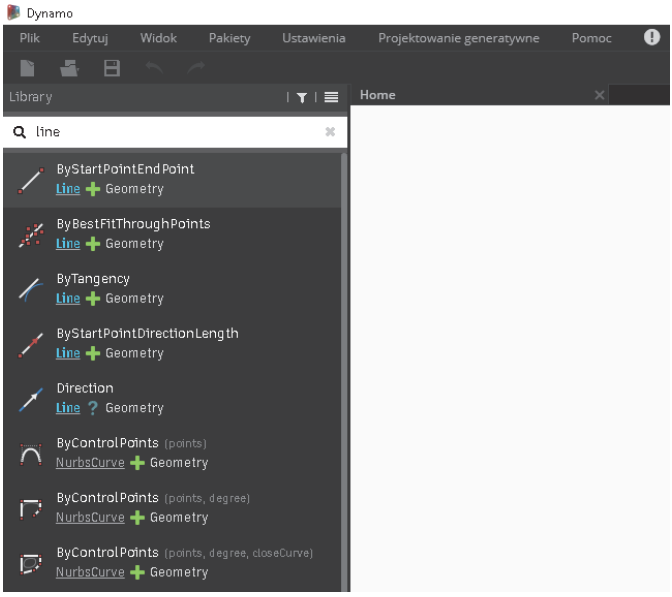
Rys. 3.13. Lokalizacja węzła *Point* w bibliotece

W tym przypadku hierarchia biblioteki jest następująca:

1. *Biblioteka* – obszar interfejsu programu.
2. *Biblioteka* – zestaw kategorii powiązanych ze sobą, w tym przypadku w obszarze geometrii.
3. *Kategoria* – zestaw powiązanych ze sobą węzłów, w tym przypadku dotyczących okręgów.
4. *Podkategoria* – podział węzłów w ramach danej kategorii, pod względem kryteriów dotyczących tworzenia, operacji i zapytań.
5. *Węzeł* – węzeł obejmujący wykonanie danej operacji, dodawany do obszaru roboczego.

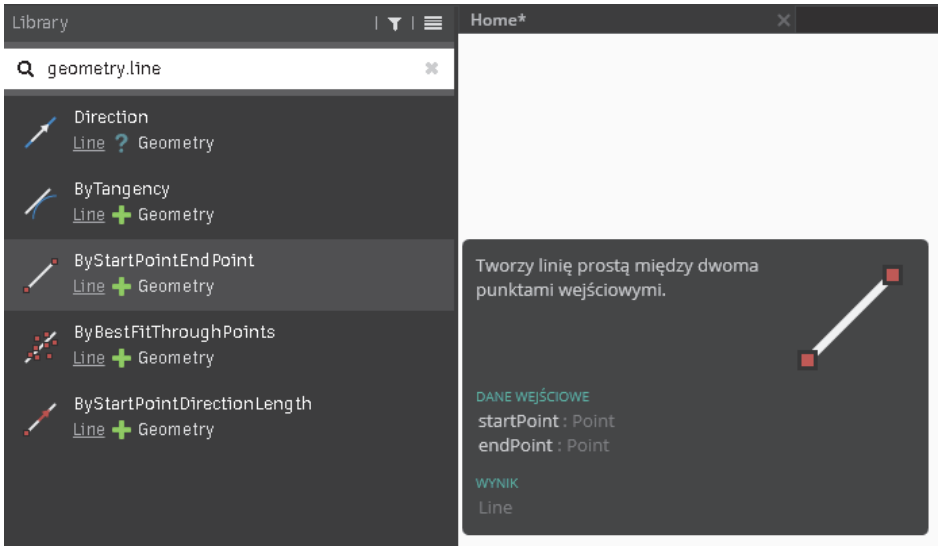
3.3.2. Wyszukiwanie węzłów

Na powyższym przykładzie widać, że biblioteka węzłów jest zhierarchizowana, a hierarchizacja oparta jest na grupowaniu węzłów zgodnie z ich specyfiką. Ma to odzwierciedlenie w nazewnictwie poszczególnych węzłów, gdzie zakodowano obszar ich działania i samo działanie. W efekcie takiego podejścia użytkownik w dość łatwy i intuicyjny sposób ma możliwość wyszukania i wczytania żądanego węzła lub innego o podobnym lub lepszym działaniu. Wydatnie pomaga w tym sposób wywoływania węzłów w programie Dynamo. Użytkownik, wpisując w polu wyszukiwania węzłów w bibliotece słowo kluczowe lub jego początek, wywołuje listę wszystkich węzłów, które są z nim powiązane. Po wpisaniu np. słowa *line* pojawia się lista wszystkich węzłów z nim stowarzyszonych, których fragment pokazano na rysunku 3.14.



Rys. 3.14. Fragment odszukanych węzłów wywołanych w bibliotece słowem *line*

W tym przypadku system zwrócił całą masę różnych węzłów z różnych kategorii i podkategorii powiązanych z komendą wywołaną słowem kluczowym *Point*. Istotna jest tu właśnie hierarchizacja organizacji węzłów w bibliotece. Użytkownik ma możliwość uwzględnić to podczas wyszukiwania węzłów. Hierarchizacja ta najczęściej lokalizuje węzeł zgodnie z systematyką biblioteka.kategoria.nazwa_węzła. Jednocześnie należy zaznaczyć, że w tym przypadku są pewne wyjątki, szczególnie jeśli chodzi o kategorie widoków i wejść. Zatem użytkownik ma możliwość wyszukiwania zaawansowanego, gdzie oprócz nazwy węzła możliwe jest ograniczenie wyszukiwania do konkretnych kategorii. W takim przypadku zapis hierarchiczny obejmuje nazwę węzła i nazwy kategorii podkategorii, stosując oddzielenie poszczególnych elementów za pomocą kropek. Zawężenie takiego poszukiwania zwraca różne wyniki, co pokazano przykładowo na rysunku 3.15.



Rys. 3.15. Wyniki wyszukiwania w bibliotece węzłów

Użytkownik ma więc możliwość wyszukiwania węzłów, stosując następujące hierarchizację i składnię:

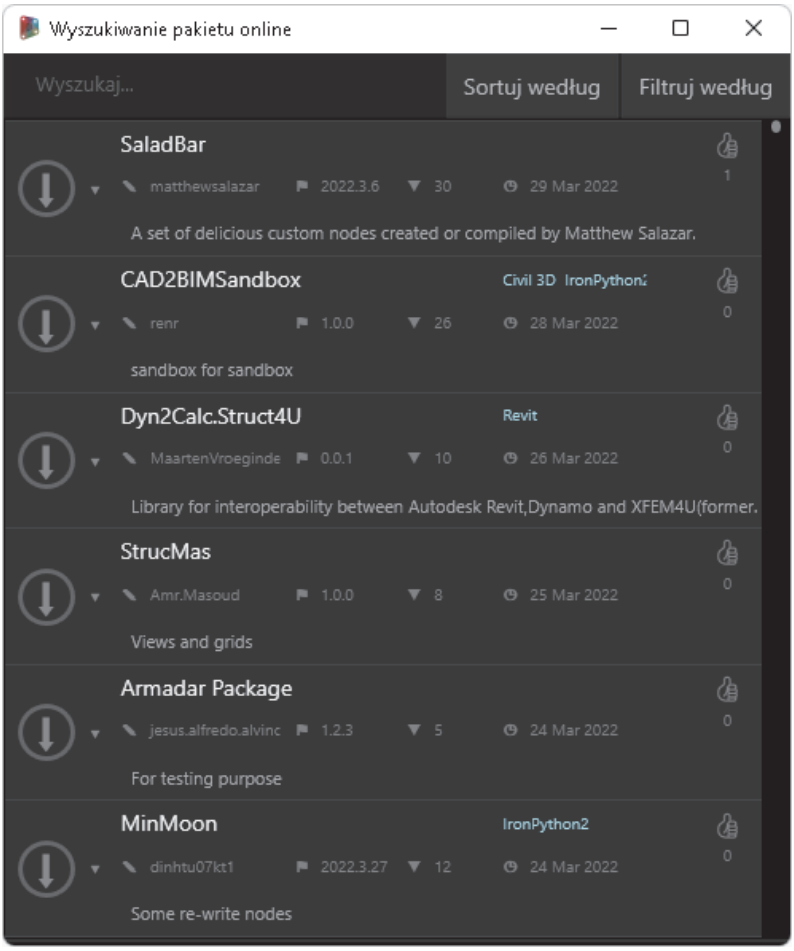
- biblioteka.kategoria.nazwa_węzła,
- kategoria.nazwa_węzła,
- nazwa węzła,
- słowo kluczowe.

Takie rozszerzenie zakresu wyszukiwania powoduje oczywiście wydłużenie listy pasujących do ustawień filtrowania węzłów.

Użytkownik musi również zwracać uwagę na podobieństwo w nazewnictwie węzłów oraz różnice w ich przynależności do poszczególnych kategorii. W sytuacji wątpliwości każdorazowo należy dokładnie sprawdzić funkcjonalność i kategorie danego węzła, aby uniknąć jego nieprawidłowego działania.

3.3.3. Pakiety węzłów

W systemie Dynamo przygotowano setki węzłów, za pomocą których można przygotować skrypty o bardzo zróżnicowanej i szerokiej funkcjonalności. W oczywisty sposób wymagają one od projektanta programisty wyobraźni, co do koncepcji przepływu danych pomiędzy wykorzystanymi w nich węzłach oraz wykonania pewnej pracy nad skryptem. Nieocenioną pomocą są tutaj pakiety węzłów, które są przygotowywane i przeznaczone specjalnie do konkretnych zastosowań. Użytkownik ma możliwość pobrania danego pakietu bezpośrednio w środowisku Dynamo (rys. 3.16) z przestrzeni internetu, jak i doinstalowania go zewnętrznie, co jest już bardziej skomplikowane.



Rys. 3.16. Opcja wyszukiwania pakietów online dostępna bezpośrednio w środowisku Dynamo

4 PODSTAWOWE ELEMENTY SKRYPTU WIZUALNEGO DYNAMO

W niniejszym rozdziale przedstawiono informacje o podstawowych i najważniejszych komponentach, z których składa się skrypt wizualny budowany w środowisku Dynamo. Oparto się na dokumentacji środowiska Dynamo [86].

4.1. PRZEGLĄD PODSTAWOWYCH I NAJCZĘŚCIEJ UŻYWANYCH WĘZŁÓW

Jak w każdym języku, nie tylko programowania, pewne jego elementy, czyli w przypadku systemu Dynamo węzły, występują częściej od innych. Niektóre węzły choć zasadniczo od siebie różne, są podstawowe i stanowią z jednej strony elementy niezbędne, np. „pojemniki na dane", lub pozwalają na podstawowe i często zachodzące operacje na danych.

Standardowa instalacja Dynamo zawiera w sobie bibliotekę węzłów, opisaną w poprzednim rozdziale. Znajdują tam się wszystkie podstawowe węzły, niezbędne do zbudowania skryptu. Użytkownik ma możliwość doinstalowania dodatkowych pakietów bibliotek węzłów. Możliwość ta, tak naprawdę bardzo często ma charakter obligatoryjny, bo przeprowadzenie pewnych operacji na danych w sposób pierwotny, tj. na zasadzie budowy podskryptów, jest po prostu zbędną pracą. Procedury te są dostępne właśnie w pakietach bibliotek węzłów, które, co istotne, są cały czas aktualizowane, a co najważniejsze poszerzane o nowe i znacząco rozbudowywane.

Poniżej przedstawiono podstawowe, najważniejsze i najczęściej używane w Dynamo węzły pochodzące z biblioteki standardowej. Jak już wspomniano, w surowej instalacji Dynamo użytkownik ma do dyspozycji setki węzłów, za pomocą których można zbudować bardzo skomplikowane skrypty. W kolejności skupiono się na kluczowych węzłach *Input* niezbędnych do określenia wejściowych parametrów programu, podglądzie działania węzła (fragmentu programu) *Watch* oraz szczególnie użytecznym elastycznym węźle typu *Code Block*, za pomocą którego można definiować parametry wyjściowe lub przeprowadzać operacje na listach.

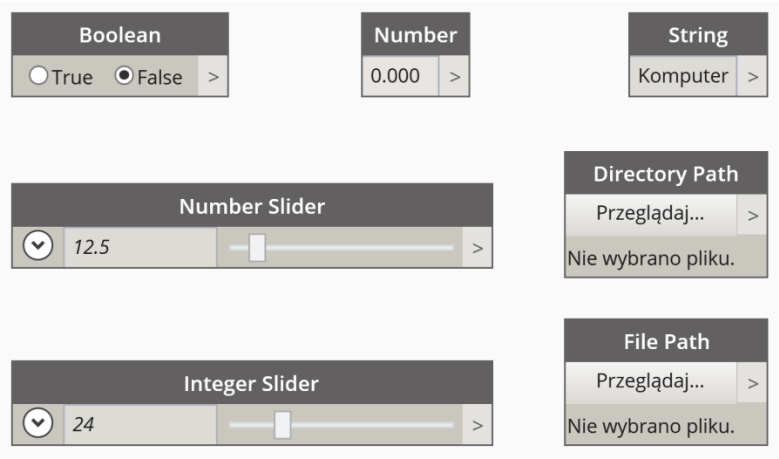
4.1.1. Wejściowe węzły danych

Za pomocą skryptów wizualnych, jak już wielokrotnie wspominano, przeprowadzane są rozmaite i ciągłe operacje na danych, o różnym charakterze. Tym samym węzły przeznaczone do wprowadzania, przechowywania i innych podobnych operacji na danych są kluczowe w tym zakresie.

Podstawowymi w tej grupie są następujące węzły typu *Input* (wejścia):

1. *Boolean* – operacje logiczne.
2. *Number* – liczba.
3. *String* – ciąg.
4. *Number Slider* – suwak liczby.
5. *Integer Slider* – suwak liczby całkowitej.
6. *Directory Path* – ścieżka do katalogu.
7. *File Path* – ścieżka do pliku.

Zostały one zestawione na rysunku 4.1.



Rys. 4.1. Węzły wejściowe

4.1.2. Węzeł podglądu *Watch*

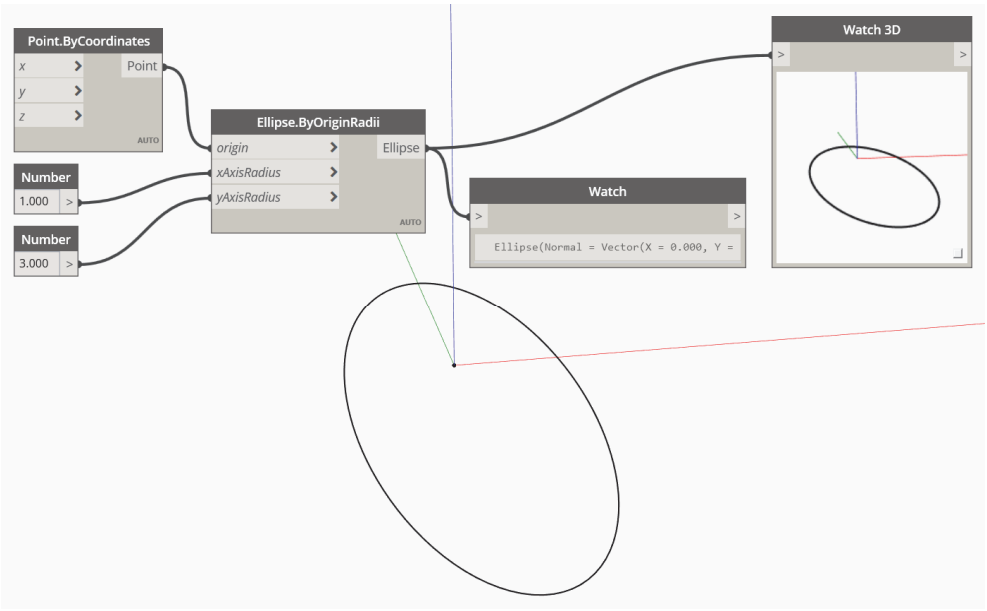
Jedną z zasadniczych zalet programowania wizualnego jest możliwość bieżącego podglądu działania całego skryptu oraz poszczególnych sekcji. Podgląd ten jest dostępny w trybie automatycznym i ręcznym. W tym pierwszym przypadku jest to niejako obserwacja działania skryptu w czasie rzeczywistym.

Bardzo przydatnym narzędziem w tym zakresie są węzły typu *Watch*. Są to tzw. węzły obserwacyjne, służące do wyświetlania efektów działania poszczególnych węzłów. Zostały one umieszczone w bibliotece w kategorii widoku. Należy nadmienić, że jest to również możliwe za pomocą podglądu danych węzła, ale działanie w takim przypadku jest nieco mniej wygodne. Za pomocą węzłów obserwacyjnych użytkownik ma możliwość stałego monitorowania i wizualizacji efektów działania skryptu.

Dostępne są dwa rodzaje węzłów typu *Watch*:

- *Watch* – pozwala na podgląd danych węzła,
- *Watch3D* – umożliwia podgląd wyników geometrii.

Działanie węzłów *Watch* i *Watch3D* pokazano na rysunku 4.2.



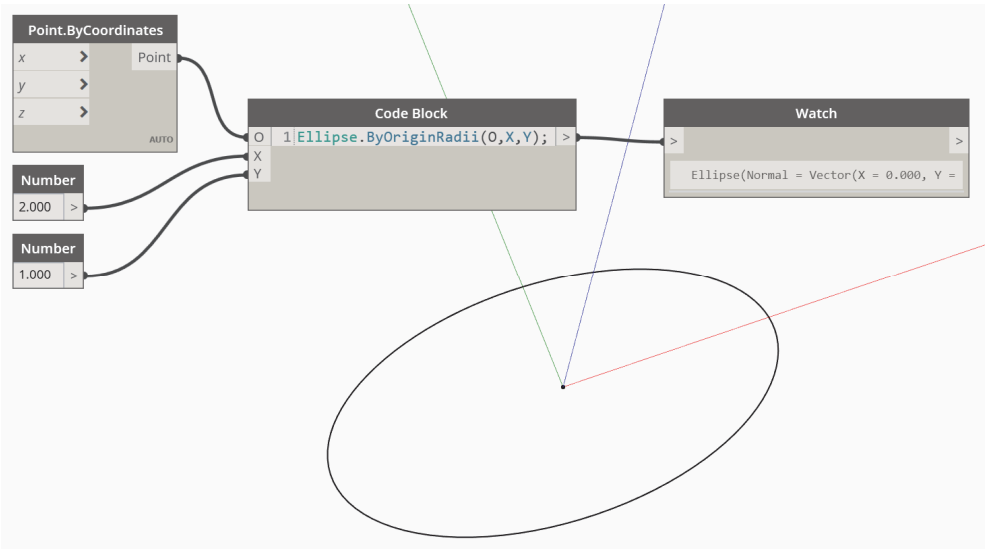
Rys. 4.2. Działanie węzłów *Watch* i *Watch3D*

4.1.3. Węzeł bloku kodu *Code Block*

Węzły typu *Code Block* to bardzo pomocne narzędzia, za pomocą których użytkownik ma możliwość definiowania rozmaitych formuł, ciągów danych itp. Jest to bardzo przydatne i wydatnie upraszcza proces przetwarzania danych, a co za tym idzie, skraca czas niezbędny do stworzenia skryptu.

Blok kodu jest definiowany w wierszach węzła *Code Block* w postaci ciągu alfanumerycznego, którego składnia jest zgodna z konwencją nazewnictwa języka DesignScript, będącym językiem tekstowym środowiska Dynamo.

Dobrym przykładem działania węzła typu *Code Block* może być skrypt służący do parametrycznej definicji elipsy. Węzeł *Code Block* zostaje wprowadzony poprzez podwójne kliknięcie w obszarze roboczym. Następnie w wierszu zostaje wpisana formuła `Ellipse.ByOriginRadii(O,X,Y);`. W celu dodania wejść *O*, *X*, *Y* należy kliknąć obszar roboczy. W dalszym kroku niezbędne jest stworzenie węzłów `Point.ByCoordinates` oraz `Number` i podłączenie ich do portów wejściowych *Code Block*. W efekcie otrzymuje się elipsę o geometrii zależnej od parametrów określonych w węzłach `Point.ByCoordinates` i `Number`. Na rysunku 4.3 pokazano przykładową składnię opisanego kodu służącą do stworzenia elipsy.



Rys. 4.3. Kod generacji elipsy

4.2. OBIEKTY SKRYPTU I OPERATORY

4.2.1. Dane

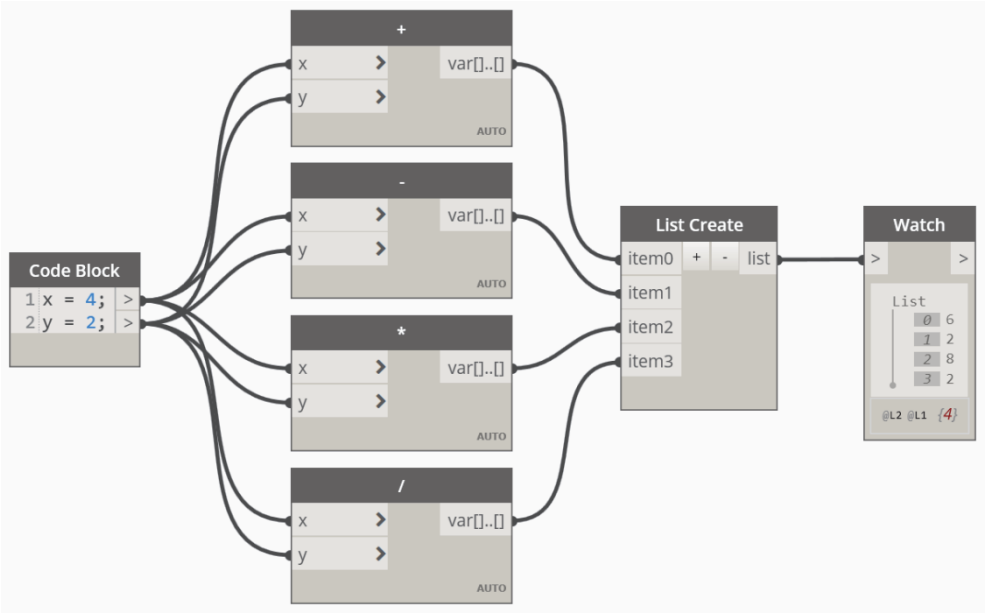
Jak w każdym innym języku programowania, tak i w środowisku Dynamo, aby uzyskać zamierzony efekt działania projektowanego skryptu, trzeba dokonać przetworzenia danych. Aby dany skrypt miał w ogóle co obrabiać, należy niejako załadować go danymi wejściowymi, które, wędrując od węzłów do węzłów, są przetwarzane, uzyskując finalnie formę danych wyjściowych. Jeśli forma ta spełnia oczekiwania twórcy, można uznać, że napisany skrypt działa poprawnie.

Dane w środowisku Dynamo określają zbiory zmiennych jakościowych lub ilościowych. Mogą to być zatem różnego rodzaju liczby, znaki lub ich ciągi, elementy geometryczne. Danymi mogą być także listy zawierające zestawy różnych elementów, np. ciągi liczbowe.

Należy wiedzieć, że jedynie określone wprowadzenie danych do węzła powoduje jego działanie. Wywołanie samego węzła powoduje na wyjściu uzyskanie funkcji, a niekoniecznie wyniku operacji.

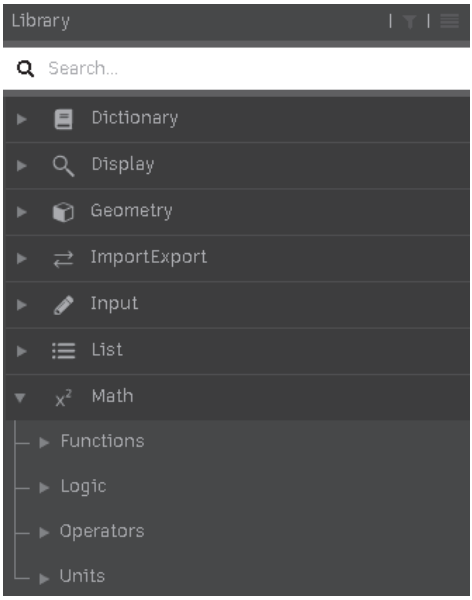
4.2.2. Operacje matematyczne

W odniesieniu do danych numerycznych, a więc zawierających liczby, podstawowymi manipulacjami na nich przeprowadzanymi są operacje matematyczne. Dynamo pozwala na przeprowadzanie elementarnych operacji typu dodawanie, odejmowanie, mnożenie, dzielenie (rys. 4.4).



Rys. 4.4. Elementarne operatory matematyczne

Składnia języka DesignScript, a więc silnika Dynamo, pozwala na przeprowadzanie bardziej skomplikowanych operacji. W bibliotece podzielono operacje na cztery podkategorie: funkcyjne, logiczne, elementarne i operacje na jednostkach (rys. 4.5).

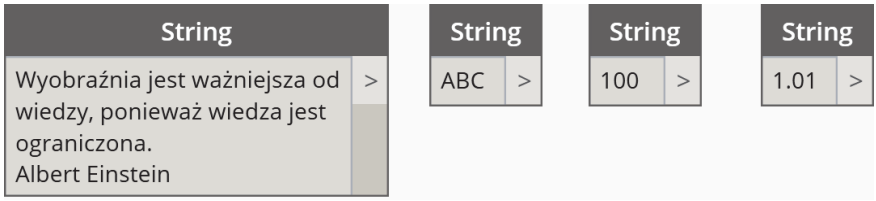


Rys. 4.5. Podkategorie operatorów matematycznych

4.2.3. Ciągi

Sekwencje znaków reprezentujących stałe literowe lub zmienne określonego typu są definiowane jako ciągi. Ciąg może również być traktowany bardziej potocznie jako tekst. Ciągi są bardzo użytecznym elementem składni samego języka DesignScript, jak i języków programowania w ogóle. Są one wykorzystywane w różnych sytuacjach i do opisu wielu elementów. Za pomocą ciągów opisywane są zestawy dokumentacji, definiowane są parametry niestandardowe, jak również są one używane do analizowania zestawów danych tekstowych.

Elementarny ciąg wywoływany jest przez węzeł *String*, który jest dostępny w kategorii *Podstawowe → Kategoria wejściowa*. Na rysunku 4.6 pokazano kilka przykładów ciągów.

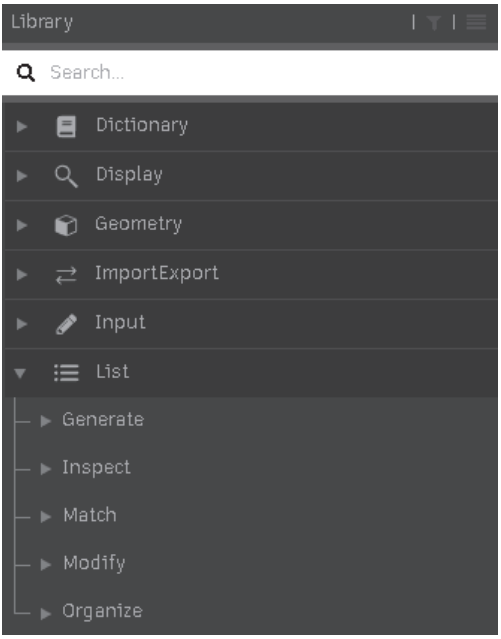


Rys. 4.6. Przykłady ciągów

Koniecznością pracy na ciągach jest manipulowanie nimi. Dotyczy to pobierania z nich pewnych danych czy w ogóle informacji, edycji i tworzenia nowych ciągów. Efektywne w tym zakresie są operacje prowadzone przy użyciu węzłów *String.Split*, *String.To.Number*, *String.Concat* czy *String.Join*.

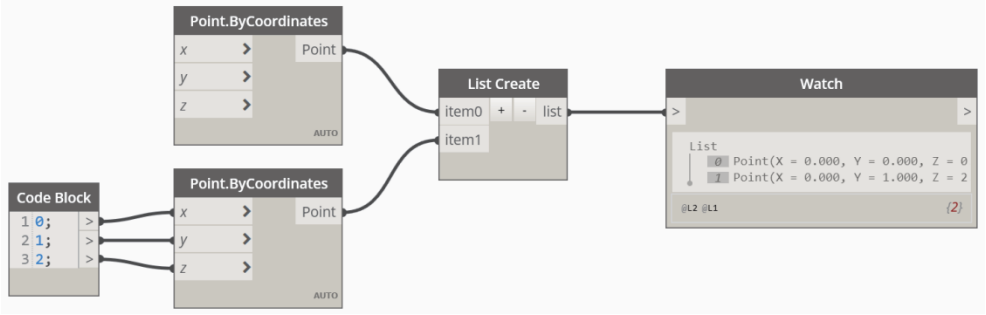
4.2.4. Listy

Tworzenie list w programie Dynamo pozwala na efektywne zarządzenie i przetwarzanie danych. Lista to nic innego jak zbiór elementów różnego typu, takich jak np. ciągi liczb czy obiekty geometryczne. W bibliotece kategoria *List*, w której znajdują się wszystkie węzły służące do tworzenia i modyfikacji list, składa się z pięciu podkategorii, pokazanych na rysunku 4.7.



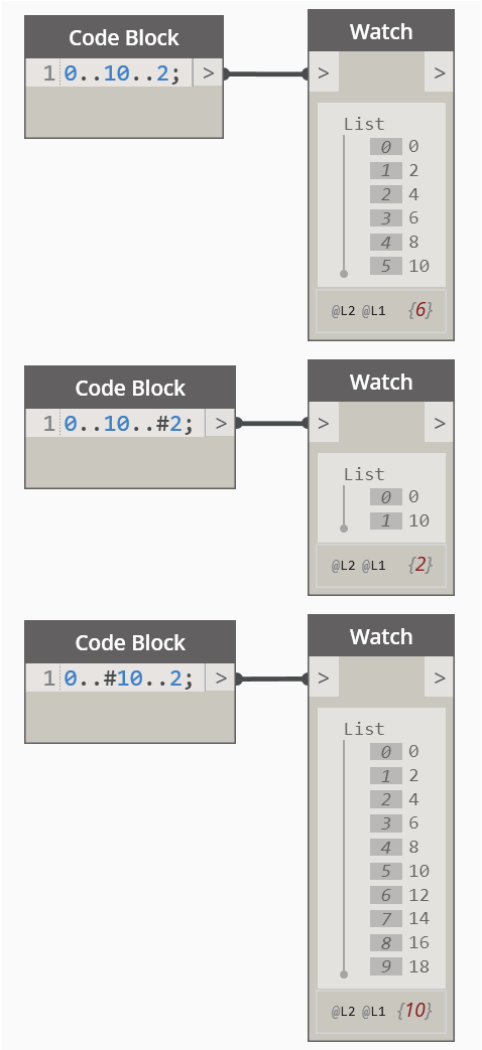
Rys. 4.7. Kategoria i podkategorie *List*

Tworzenie list w programie Dynamo pozwala na efektywne zarządzenie i przetwarzanie danych. Podstawowym węzłem, który jest chyba najczęściej stosowany, jest *List Create*, w którym można stworzyć i zintegrować nową listę z określonych danych wejściowych. Na rysunku 4.8 pokazano budowę nowej listy, w której zawarto dwa punkty.



Rys. 4.8. Przykład listy zawierającej dwa punkty

Kolejną bardzo użyteczną i często wykorzystywaną opcją jest tworzenie list danych w postaci ciągów liczb w oparciu o kodowanie DesignScript. Tematyka ta jest dość szeroka i wymaga bliższej znajomości składni tego języka. Elementarne natomiast jest tworzenie zakresów ciągów liczb w układzie pokazanym na rysunku 4.9.



Rys. 4.9. Tworzenie list za pomocą generacji ciągu elementów

Zbiory liczb wygenerowane w kolejnych przypadkach są następujące:

- 0..10..2; z zakresu od 0 do 10 wygenerowano elementy z interwałem 2,
- 0..10..#2; z zakresu od 0 do 10 wygenerowano 2 elementy,
- 0..#10..2; z zakresu od 0 wygenerowano 10 elementów z interwałem 2.

Tematyka tworzenia i zarządzania listami jest bardzo szeroka i wykracza poza zakres niniejszej publikacji, ale została dobrze przybliżona w dokumentacji programu [86].

4.3. ELEMENTARNE OBIEKTY GRAFICZNE

Programowanie wizualne znajduje rozmaite zastosowanie, co opisano szeroko we wprowadzeniu do niniejszej monografii. W odniesieniu do projektowania inżynierskiego podstawowe jest kreowanie formy, a dokładniej geometrii elementu będącego przedmiotem zainteresowania. W tym zakresie koncepcja parametryzacji geometrii jest zagadnieniem samym w sobie, co stanowi przedmiot studiów interdyscyplinarnych i kierunków rozwoju niektórych prądów, np. architektury [44-56]. Tym samym niezbędne jest określenie i zdefiniowanie filozofii tworzenia i definiowania geometrii, jaką przyjęto w środowisku Dynamo.

Geometria w środowisku Dynamo to tak naprawdę jądro tego języka programowania. Dotyczy to z jednej strony określenia i zdefiniowania podstawowych obiektów geometrycznych, z których zbudowana będzie projektowana struktura, a z drugiej strony określenia odpowiedniej hierarchii poszczególnych elementów. Niestety jest jeszcze trzecia kwestia, a mianowicie dostosowanie geometrii projektowanej struktury do geometrii standaryzowanej, tzn. funkcjonującej w innych środowiskach, z którymi Dynamo współpracuje lub w nich egzystuje. Dotyczy to nie tylko formatu cyfrowego typu DXF, CAD, RVT i innych, ale w przypadku współpracy z programami obliczeniowymi jest to związane z koncepcją i wymogami w nich określonymi. Głównie dotyczy to budowy modeli obliczeniowych wynikającej z zastosowanej metody obliczeniowej w danym programie. Najczęściej stosowaną jest metoda elementów skończonych, w której model obliczanej struktury składa się tak naprawdę z kilku typów tzw. elementów skończonych jedno-, dwu- i trójwymiarowych. Jest to pewien problem, o czym będzie wspomniane przy okazji modelowania przykładowych struktur przedstawionych w dalszej części monografii.

4.3.1. Pojęcia podstawowe i hierarchia geometrii

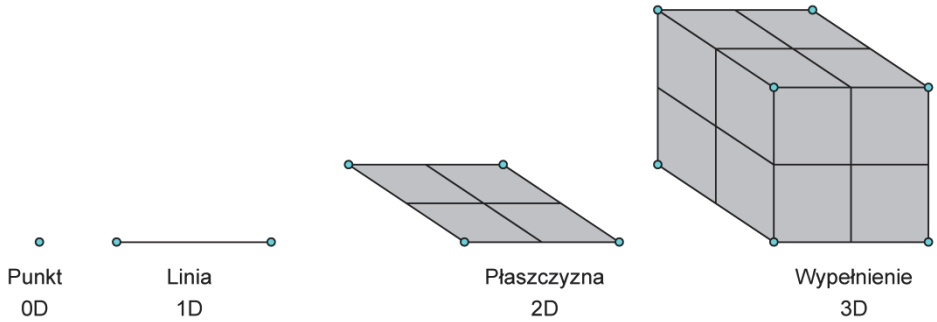
Jednym z podstawowych problemów w omawianym zakresie jest zapewnienie odpowiedniego „przetłumaczenia" geometrii na dany język programowania, a dokładniej zrozumienie jej przez algorytmy w nim funkcjonujące. To samo tyczy się programisty, który musi mieć na tyle wyobraźni, żeby ujrzeć daną geometrię w przestrzeni algorytmów i wykonywanych przez nie operacji.

Kluczowy jest tutaj abstrakcyjny sposób myślenia o geometrii, która jest opisywana mniej lub bardziej ściśle matematycznie. W konsekwencji można podać tu kilka podstawowych reguł. Tak więc geometrię należy traktować jako dane, a poszczególne elementy geometryczne są opisywane przez wzory lub bardziej skomplikowane formuły definiujące poszczególne zależności między kluczowymi elementami determinującymi kształt, rozmiar, położenie danej figury w trójwymiarowym układzie współrzędnych (najczęściej) przestrzeni modelu. Geometria jest podporządkowana układowi hierarchicznemu, gdzie pewne składowe elementów geometrycznych są sobie podporządkowane. Najprostszy ciąg jest taki, że najniżej w hierarchii są punkty, które połączone tworzą linię. Kilka stykających się ze sobą

linii pozwala na definicję powierzchni, które z kolei odpowiednio stykając się ze sobą, pozwalają na wytworzenie bryły. Ostatnim wyróżnikiem geometrii jest jej cecha – opisuje ona zarówno cały element, jak i jego części. Można to wykorzystać, choćby generując rozkład punktów rozmieszczonych po krzywej.

Obrazowanie rzeczywistego, fizycznego obiektu to operacja modelowania na pewnym poziomie abstrakcji. Przyjmując matematyczną hierarchizację podstawowych obiektów składowych (*primitives*), coraz bardziej złożone obiekty składają się z elementów *n*-wymiarowych takich jak (rys. 4.10):

- punkt,
- linia,
- płaszczyzna,
- wypełnienie.



Rys. 4.10. Podstawowe obiekty geometryczne

Obiekty te można zdefiniować w następujący sposób:

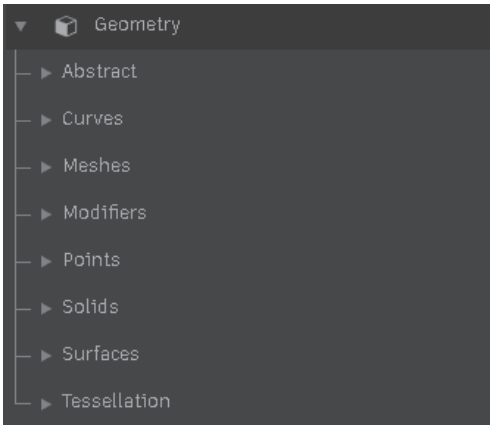
1. *Punkt* – definiowany za pomocą współrzędnych, brak wymiarów (0D).
2. *Linia* – definiowana przez dwa punkty, ma jeden wymiar (1D).
3. *Płaszczyzna* – definiowana dwiema liniami, ma dwa wymiary (2D).
4. *Prostopadłościan* – definiowany za pomocą dwóch płaszczyzn, ma trzy wymiary (3D).

Tym samym powyższa hierarchizacja oparta jest na przestrzeni euklidesowej i poprawnie opisuje geometrię trójwymiarową składającą się z elementów pokazanych na rysunku 4.10. Problematyczna jest natomiast kategoryzacja uwzględniająca obiekty geometryczne o innej geometrii, np. zakrzywionej, oraz cechy abstrakcyjne, takie jak wzajemne powiązania pomiędzy obiektami, orientacja czy funkcjonowanie wektorów. W tabeli 4.1 pokazano bardziej szczegółową kategoryzację, która dzieli dane na dwa typy, abstrakcyjne i geometryczne, starając się uwzględnić wyżej wymienione cechy i zjawiska.

Tabela 4.1. Szczegółowa kategoryzacja typów danych w środowisku Dynamo

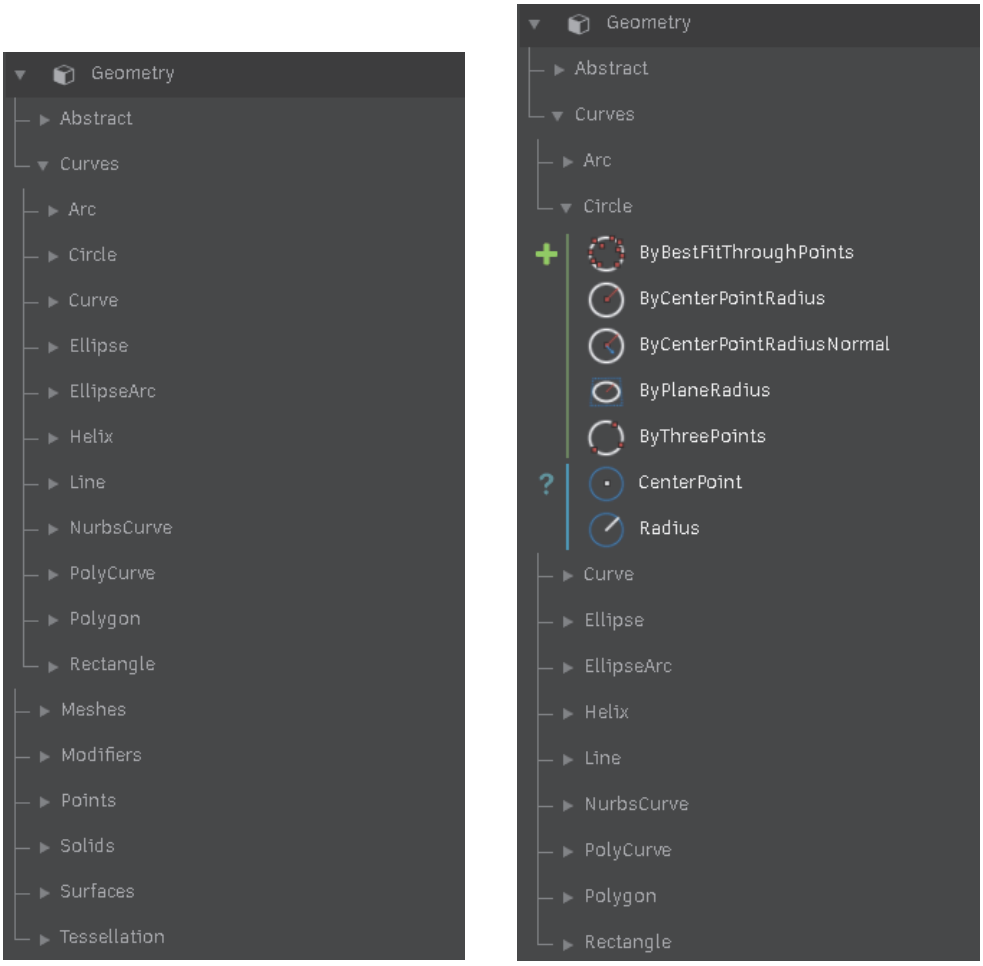
Typ	Element	Funkcja	Obiekt
Typy abstrakcyjne	Wektor	Definiuje położenie i orientację	– wektor – płaszczyzna – układ współrzędnych
	Topologia	Definiuje relacje	– wierzchołek – krawędź – powierzchnia
Typy geometrii	Punkt	Element modelu	– współrzędna X, Y, Z – współrzędna U, V
	Łuk	Element modelu	– linia – wielobok – łuk – okrąg – elipsa – krzywa NURBS – krzywa PolyCurve
	Powierzchnia	Element modelu	– powierzchnia NURBS – powierzchnia PolySurface
	Wypełnienie	Element modelu	– prostopadłościan – sfera – stożek – walec
	Siatka	Element modelu	– siatka

W efekcie w środowisku Dynamo przyjęto określoną kategoryzację i hierarchię węzłów definiujących ogólnie traktowaną geometrię (rys. 4.11).



Rys. 4.11. Hierarchizacja obiektów geometrycznych

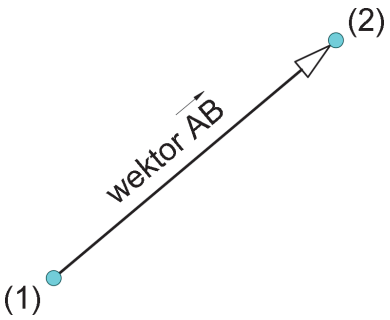
Węzły te są uporządkowane w bibliotece alfabetycznie, a nie hierarchicznie. Pokazano je na rysunku 4.12.



Rys. 4.12. Hierarchizacja podkategorii i węzłów w kategorii *Geometria*

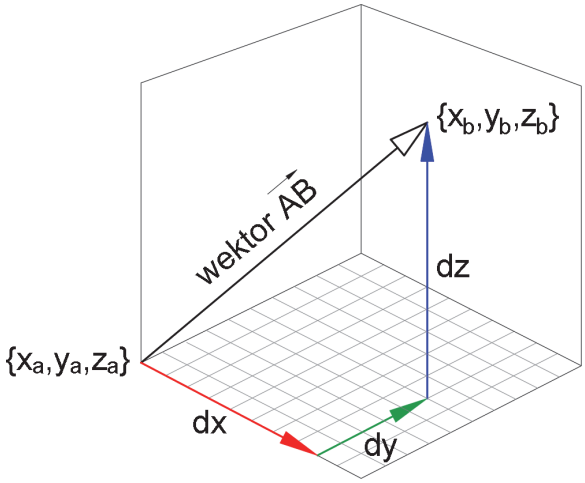
4.3.2. Wektor

Wszystkie obiekty geometryczne funkcjonują w przestrzeni roboczej środowiska Dynamo w trójwymiarowej przestrzeni euklidesowej określonej układem współrzędnych *XYZ*. Obiektem określającym, a w pewnych sytuacjach niejako sterującym położeniem elementów geometrycznych jest wektor. Tak jak w matematyce wektory są obiektami geometrycznymi, które są definiowane (opisane) przez wielkości takie jak moduł, kierunek i zwrot. W środowisku Dynamo definiuje się wektory zaczepione z uwagi na określenie ich początku i końca. Początek wektora (1) to punkt jego zaczepienia, a koniec (2) to wierzchołek lub zwrot (rys. 4.13).



Rys. 4.13. Wektor

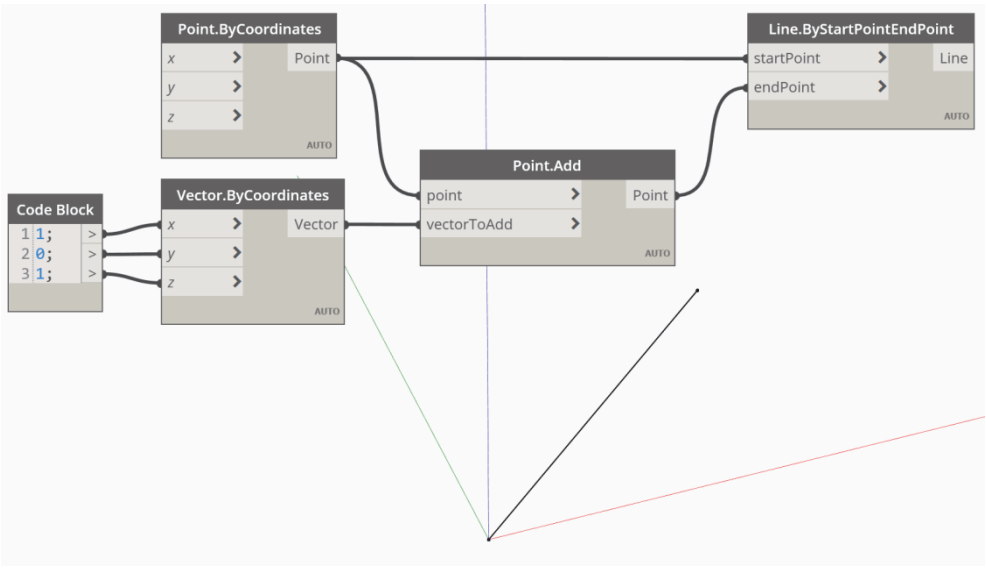
Wektory nie są ściśle elementami geometrycznymi z uwagi na cechy abstrakcyjne (por. tabela 4.1). Tym samym określają one wielkość.



Rys. 4.14. Definicja wektora

W środowisku Dynamo wektory definiowane są listą wartości, które opisują względną różnicę w pozycji, która odpowiada pojęciu „kierunku”. Wektor AB pokazany na rysunku 4.14 opisany jest współrzędnymi {*dx*, *dy*, *dz*}, gdzie wielkości *di* stanowią składowe długości z rozbiciem na poszczególne współrzędne, tj. $dx = x_b - x_a$, $dy = y_b - y_a$, $dz = y_b - y_a$.

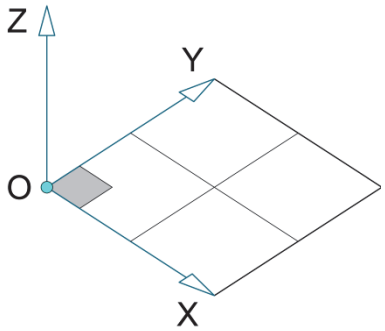
Zgodnie z nomenklaturą systemu Dynamo wektory to „abstrakcyjne elementy pomocnicze”. Należy również zauważyć, że w podglądzie tła nie ma możliwości wywołania wizualizacji wektorów (rys. 4.15).



Rys. 4.15. Wektor zdefiniowany w skrypcie

4.3.3. Płaszczyzna

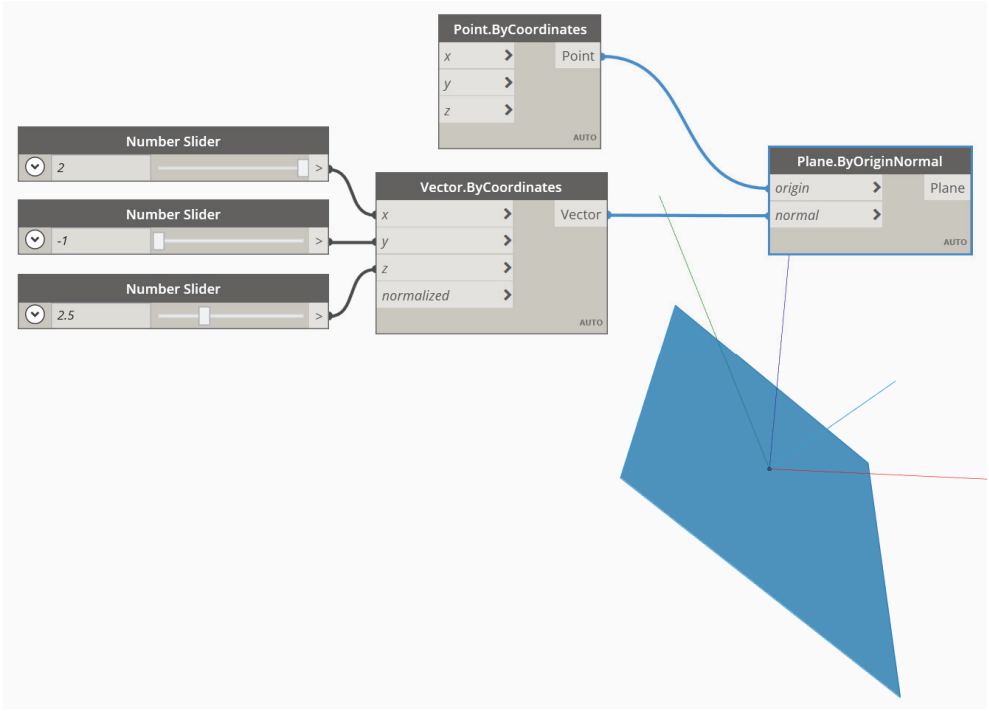
Kolejnym abstrakcyjnym elementem geometrycznym, który odgrywa podstawową rolę w środowisku Dynamo, jest płaszczyzna. Jest to nic innego jak płaski i dwuwymiarowy element. Cechą charakterystyczną płaszczyzny jest to, że jest ona zorientowana względem początku, który określa jej położenie. Płaszczyzna jest również definiowana poprzez dwa prostopadłe do siebie kierunki *X* i *Y*, które określają jej obszar. Nie ma ona grubości (głębokości), natomiast określa się kierunek normalny do niej opisany jako kierunek *Z* (rys. 4.16).



Rys. 4.16. Oznaczenie kierunków płaszczyzny

Początek płaszczyzny zaznaczony jest ciemniejszym kwadratem. Płaszczyzny są obiektami, które można utożsamiać z obiektami używanymi w innych środowi-

skach modelowania BIM, jak poziomy czy rzuty (rzutnie). Mogą być one zorientowane ortogonalnie w układzie *XYZ* i wtedy definiowane są kierunkami *XY*, *YX* i *XZ*. Możliwe oczywiście jest ich dowolne ułożenie w przestrzeni trójwymiarowej i wtedy są one ustawione pod pewnym kątem w stosunku do układu osi *XYZ*. Pokazano to przykładowo na rysunku 4.17.

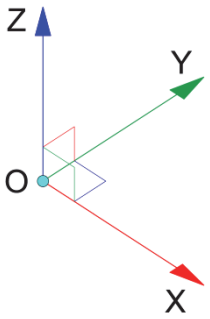


Rys. 4.17. Definicja płaszczyzny

4.3.4. Układ współrzędnych

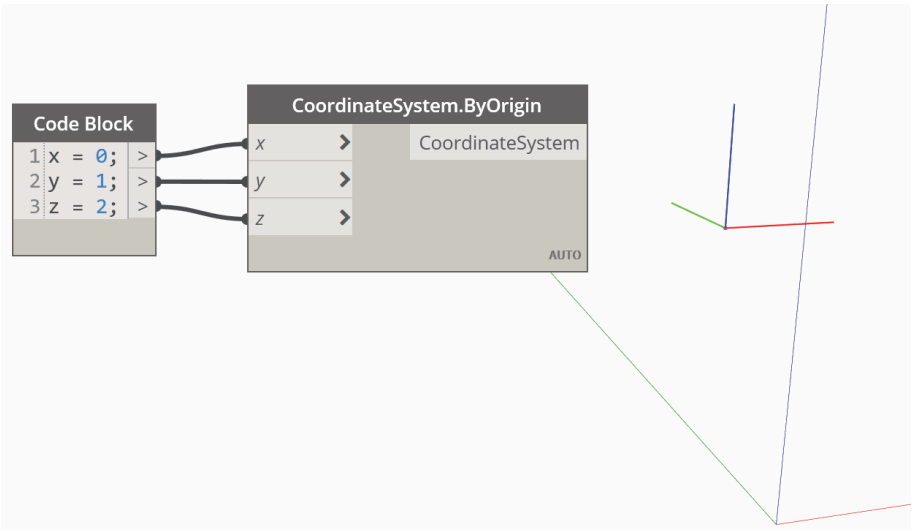
Układ współrzędnych to kolejny z abstrakcyjnych elementów geometrycznych funkcjonujących w środowisku Dynamo. Tak jak w przypadku klasycznej geometrii euklidesowej można określić kilka układów współrzędnych.

Najczęściej stosowany jest układ ortogonalny *XYZ*, określony przez punkt początkowy i wzajemnie prostopadłe osie *X*, *Y* i *Z* (rys. 4.18).



Rys. 4.18. Ortogonalny układ współrzędnych

W zasadzie elementy te odpowiadają tym samym elementom definiującym obiekt typu płaszczyzna. Tego typu układ współrzędnych należy traktować tak samo jak w klasycznej geometrii, ale też i w środowiskach CAD. Układ ortogonalny określa nam początek odniesienia i ułożenie poszczególnych osi *XYZ* (rys. 4.19).



Rys. 4.19. Określenie początku ortogonalnego układu współrzędnych

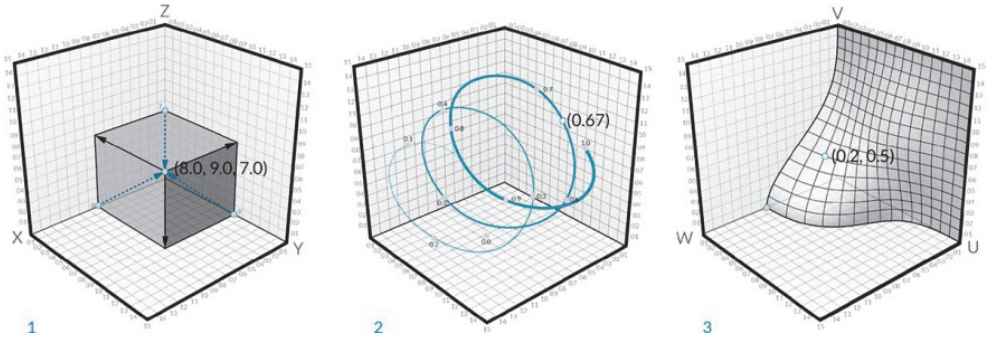
Tak jak w klasycznej geometrii tak w środowisku Dynamo dostępne są inne układy współrzędnych, tj. układy walcowy i sferyczny. Są one definiowane odpowiednio przez węzły *CoordinateSystem.ByCylindricalCoordinates* i *CoordinateSystem.BySphericalCoordinates*.

Układy współrzędnych w Dynamo są zorientowane (umieszczane) w przestrzeni. Przykładowo układ ortogonalny (1) pokazany na rysunku 4.19 został umieszczony w pewnym oddaleniu od punktu początkowego przestrzeni definiowanego jako „(0, 0, 0)”. Jego osie są oznaczone kolorami, gdzie oś *X* jest czerwona, oś *Y* zielona, a oś *Z* niebieska.

4.3.5. Punkt

Do definicji praktycznie wszystkich obiektów geometrycznych, zarówno tych najbardziej elementarnych, jak i tych bardziej skomplikowanych niezbędne są m.in. punkty. Odcinek posiada początek i koniec określony punktami, a krzywa może mieć dodatkowo punkt określający jej wygięcie. Tym samym punkty to najbardziej elementarne obiekty geometryczne w środowisku Dynamo.

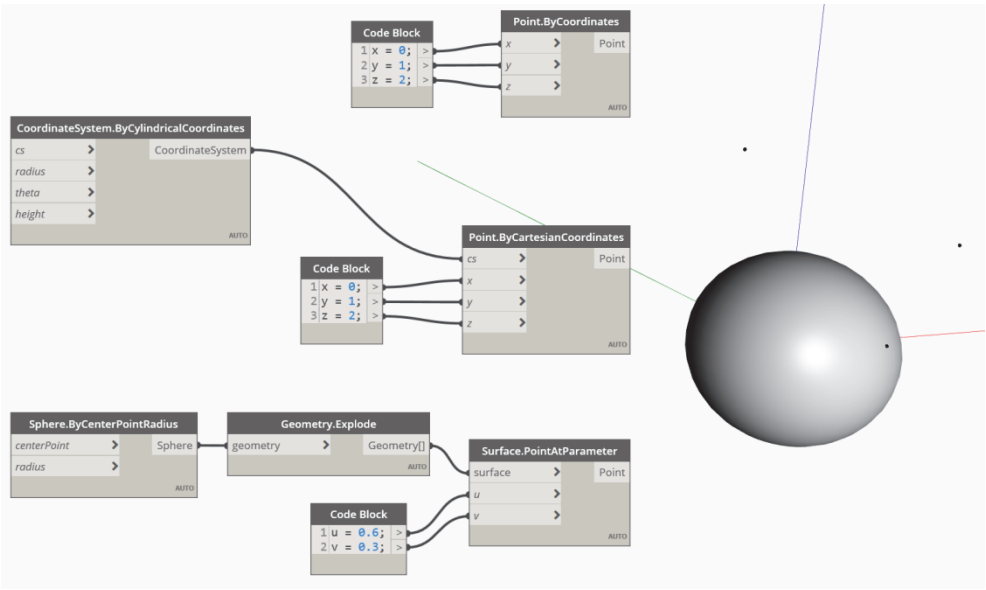
Zasadniczo punkt definiowany jest za pomocą współrzędnych. W zależności od przyjętego układu mogą to być trzy wartości odpowiednich współrzędnych *XYZ*, jak w przypadku najczęściej stosowanego kartezjańskiego układu współrzędnych. W takim przypadku współrzędne punktu określone są jako (*X*, *Y*, *Z*). Istnieje również możliwość operowania i funkcjonowania punktu w dwuwymiarowym układzie współrzędnych. Definiując punkt na płaszczyźnie, jest on opisany współrzędnymi (*X*, *Y*), natomiast na powierzchni jako (*U*, *V*). W zależności od zastosowanego układu można operować różnymi współrzędnymi, pokazanymi na rysunku 4.20.



Rys. 4.20. Definicja punktu w wielowymiarowych układach współrzędnych: (1) euklidesowy układ współrzędnych: (*X*, *Y*, *Z*); (2) układ współrzędnych z parametrem krzywej: (*t*); (3) układ współrzędnych z parametrami powierzchni: (*U*, *V*) [86]

Stosując układy dwuwymiarowe, tworzy się sytuacja niejako lokalnych układów współrzędnych, gdzie punkty definiowane są w odniesieniu do obiektu, w przestrzeni którego funkcjonują. Współrzędne te mogą być oczywiście transformowane do układu globalnego *XYZ*.

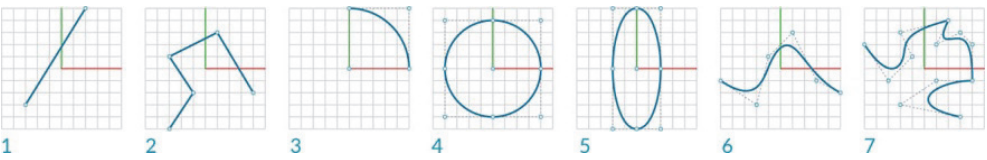
Użytkownik ma możliwość pracy w innych układach współrzędnych takich jak walcowy czy sferyczny. Przykład definicji punktu w tych układach pokazano na rysunku 4.21.



Rys. 4.21. Definicja punktu w różnych układach współrzędnych: globalny układ współrzędnych *XYZ*; układ współrzędnych walcowych; układ współrzędnych sferycznych

4.3.6. Krzywe

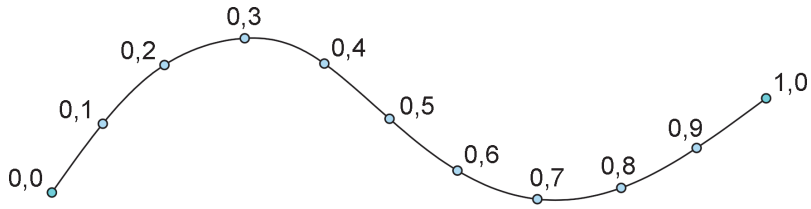
System Dynamo oferuje szeroki zestaw elementów typu krzywe. Są one niczym innym jak elementami jednowymiarowymi, za pomocą których można zdefiniować dowolny element czy obiekt geometryczny składający się z tego typu geometrii. Nazwa *krzywe* zawiera w sobie elementy niekoniecznie zakrzywione, chociażby takie jak linia czy linia łamana, za pomocą której można zbudować np. wielobok. Pełny katalog krzywych zawiera obiekty pokazane na rysunku 4.22 i są to *linia*, *polilinia*, *łuk*, *okrąg*, *elipsa*, *krzywa NURBS*, *krzywa PolyCurve*.



Rys. 4.22. Krzywe: (1) linia, (2) polilinia, (3) łuk, (4) okrąg, (5) elipsa, (6) krzywa NURBS, (7) krzywa PolyCurve [86]

Krzywe w środowisku Dynamo to geometryczne elementy jednowymiarowe, które można opisać funkcją (funkcjami) określoną jednym parametrem. Linia, a dokładniej odcinek opisany może być funkcją $y = x$, gdzie $x = 2t$. Tym samym $y = 2t$. Dla innych obiektów geometrycznych (por. rys. 4.22) konieczne jest wprowadzanie funkcji wyższego rzędu, jednakże zawsze można w nich zastosować

jeden parametr *t*. Właśnie jego wartością można sterować, a tym samym definiować i elastycznie modyfikować geometrię modelowanego obiektu. Zbiór wartości parametru *t* (domena) w środowisku Dynamo traktowany jest niejako względnie i wynosi $t < 0 ; 1>$. Przykładem może być krzywa pokazana na rysunku 4.23.

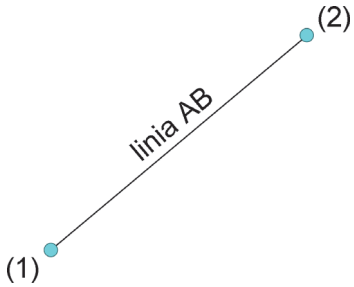


Rys. 4.23. Krzywa opisana parametrem *t*

Oprócz parametru *t* krzywe opisane są punktami początkowymi i końcowymi oraz innymi cechami zależnymi od danego typu krzywej. Istotne są tutaj pośrednie punkty kontrolne, które pozwalają na dokładniejsze dostosowanie geometrii elementu.

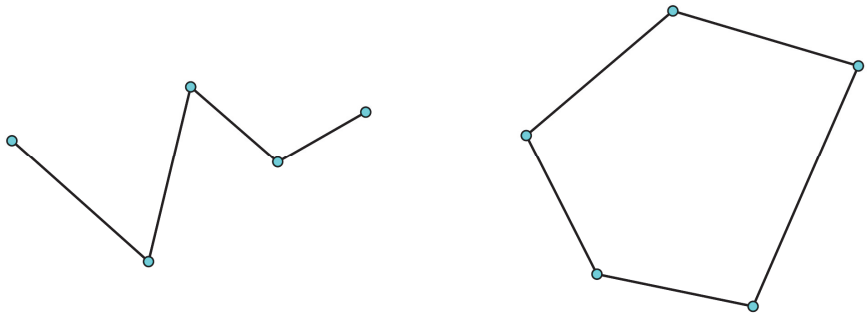
4.3.7. Linie

Najbardziej elementarnym i prymitywnym obiektem w rodzinie krzywych jest linia. Jest ona definiowana punktem początkowym i końcowym (rys. 4.24).



Rys. 4.24. Linia

Z kilku połączonych ze sobą linii można zbudować obiekty typu *polilinia* (rys. 4.25). W tego typu krzywej łamanej punkty styku tworzą swego rodzaju punkty kontrolne, tak charakterystyczne i niezbędne w krzywych „wyższego" rzędu. Edycja punktu kontrolnego powoduje zmianę jego położenia i co za tym idzie także zmianę położenia segmentów polilinii stykających się w nim.

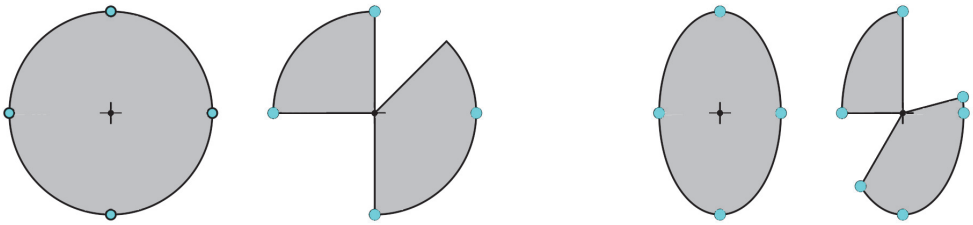


Rys. 4.25. Przykłady polilinii otwartej i zamkniętej

Jeśli początkowa i końcowa linia segmentu polilinii łączą się ze sobą, polilinia stanowi wielobok (rys. 4.25).

4.3.8. Łuk, okrąg, łuk eliptyczny, elipsa

Obiektami wyższego rzędu w porównaniu do linii i polilinii są okręgi i elipsy oraz ich fragmenty, czyli łuki (rys. 4.26).



Rys. 4.26. Okrąg, elipsa i łuki

Wyższy rząd związany jest z bardziej skomplikowanym opisem matematycznym tego typu elementów geometrycznych, których równania mają postać nie funkcji liniowej, ale bardziej skomplikowanej funkcji kwadratowej. W przypadku okręgu jest to funkcja postaci $(x - x_0)^2 + (y - y_0)^2 = r^2$, podczas gdy elipsę opisuje funkcja $(x - x_0)^2/a^2 + (y - y_0)^2/b^2 = 1$. Jak widać, do opisu tego typu obiektów potrzeba nieco więcej parametrów, bo w przypadku elipsy oprócz jej środka należy zdefiniować dwie połówki jej odcinków osiowych. W przypadku łuków będących wycinkami okręgu lub elipsy dodatkowo trzeba określić punkty początkowe i końcowe.

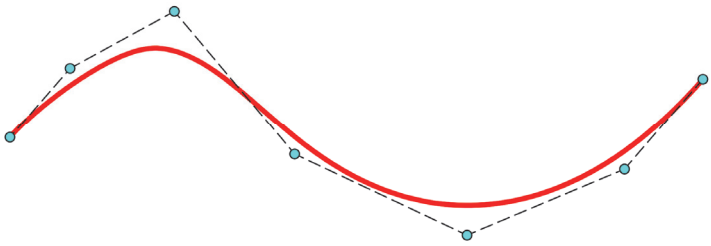
4.3.9. Krzywe NURBS i PolyCurve

Choć antyczna geometria oparta jest przede wszystkim na krzywiznach będących wycinkami okręgu czy elipsy, to oczywiście inne rodzaje krzywizn również w niej występują. Obecnie trudno chyba sobie wyobrazić kształtowanie formy

wielu różnorakich obiektów budowlanych czy konstrukcji, które spełniałyby rygorystyczne wymogi klasycznego kanonu piękna opartego na symetrii i skończonemu katalogowi krzywizn. Od lat różni architekci szukali i szukają nowych form, które uwolnione byłyby z tych wymogów i ograniczeń. Trendem, który funkcjonuje do dnia dzisiejszego i wcale nie jest w odwrocie, jest odniesienie do form organicznych. Za prekursora i propagatora można tu przywołać Antonio Gaudiego [87], ale nie można zapominać o innych twórcach, jak choćby Santiago Caltrava [88] czy Zaha Hadid [89], którzy w nie tak dawnym czasie wytyczyli niejako trendy w tym zakresie. Obecnie obserwuje się coraz bardziej futurystyczne formy, które nadają kształty budynkom, mostom, kładkom dla pieszych czy rozmaitym konstrukcjom inżynierskim. Jedną z cech wspólnych jest ich organiczność, wydawałoby się całkowicie nieujarzmiona geometrycznie i matematycznie. Jest to oczywiście złudzenie, bo w zasadzie każdy kształt i geometrię można opisać za pomocą mniej lub bardziej wyrafinowanej funkcji matematycznej.

Podstawowe są tu kształty krzywych tzw. splajny (*spline*). Formalnie obiekty tego typu określane są jako krzywe B-sklejane (ang. *B-spline*) i stanowią połączone ze sobą krzywe różnego stopnia.

Obiektami graficznymi, za pomocą których można modelować zarówno najbardziej wyrafinowane krzywizny, obiekty symetryczne, jak choćby koło, elipsę, ale też obiekty „niekrzywe", czyli linie, są krzywe typu NURBS. Te z kolei są rozwinięciem krzywych typu splajn, ale w przestrzeni dwuwymiarowej. NURBS to akronim z języka angielskiego terminu *Non-Uniform Rational B-Spline*. Sam termin NURBS jest stosowany do określania dwóch rodzajów obiektów: krzywych oraz powierzchni. Krzywe NURBS są bardzo wygodne i efektywne w użytku. Z uwagi na znaczną liczbę punktów kontrolnych, które je definiują, pozwalają na dokładne określenie geometrii oraz elastyczność w jej modyfikacji (rys. 4.27).

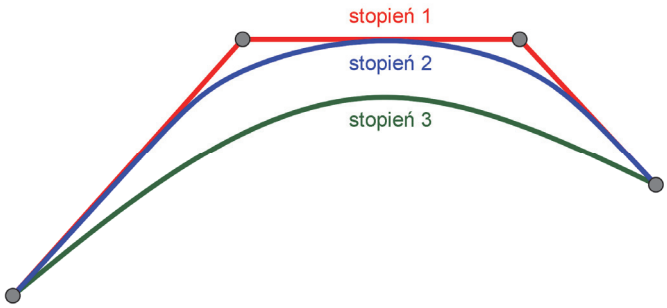


Rys. 4.27. Krzywa typu NURBS

Sterowanie krzywymi NURBS w systemie Dynamo (i zasadniczo w ogóle) jest oparte na kilku kluczowych parametrach omówionych poniżej.

Kształt krzywej określony jest punktami, które są rozłożone w charakterystycznych miejscach. Ich położenie wynika ściśle z formuły opisującej dany fragment (odcinek) krzywej. Istotne są tutaj tzw. punkty kontrolne, które nadają określoną wypukłość i przebieg danego fragmentu. Istnieje również pewien kluczowy para-

metr, wyróżnik krzywej NURBS określany mianem jej stopnia. Określa on zakres wpływu, jaki na daną krzywą mają punkty kontrolne. Wpływ ten wzrasta wraz ze wzrostem stopnia, który przybiera postać liczy całkowitej, najczęściej z zakresu od 1 do 5. Najniższy stopień 1 określa linie NURBS i polilinie. Krzywe swobodne z kolei cechują się wyższymi stopniami, najczęściej 3 lub 5. Na rysunku 4.28 pokazano, jak stopień krzywej typu NURBS wpływa na jej kształt.



Rys. 4.28. Krzywe NURBS o stopniach 1, 2 i 3

Jak widać, wraz ze wzrostem stopnia krzywej NURBS wrasta liczba punktów kontrolnych. Punkty kontrolne można zdefiniować jako listy punktów o długościach wynoszących o jeden więcej niż stopień danej krzywej. W celu zmiany kształtu danej krzywej wystarczy zmienić położenie danego punktu kontrolnego.

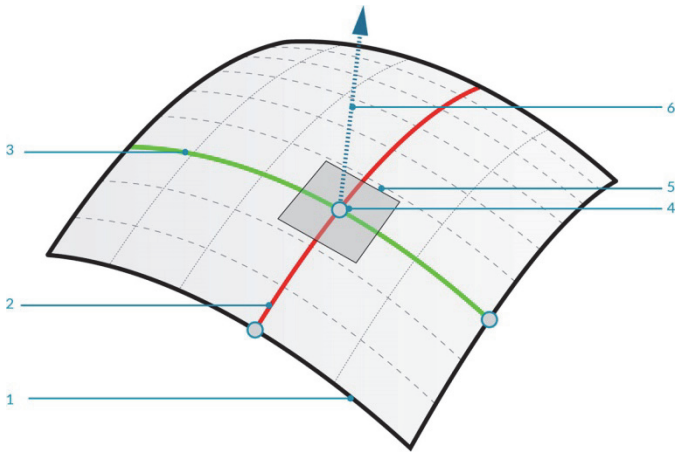
Innym parametrem opisującym krzywe NURBS są wagi, będące liczbami skojarzonymi z punktami kontrolnymi. Krzywa jest klasyfikowana jako niewymierna, gdy wszystkie punkty kontrolne krzywej mają tę samą wagę. Najczęściej jest to waga 1, a większość krzywych NURBS jest właśnie krzywymi niewymiernymi. W przypadku gdy wagi są różne krzywe określa się jako wymierne.

Z kolei węzłami określane są listy liczb równych *stopień* + N – 1, a wielkość N opisuje liczbę punktów kontrolnych. Węzły pełnią rolę parametrów sterujących wpływem punktów kontrolnych na geometrię danej krzywej NURBS, razem z jej wagami. Węzły są używane do tworzenia punktów podziału danej krzywej w pewnych jej obszarach.

4.3.10. Powierzchnie

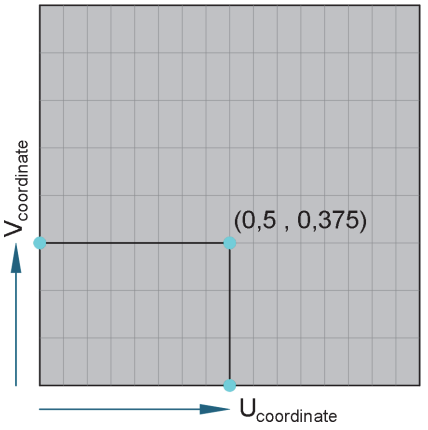
Elementami wyższego rzędu, jeśli chodzi o wymiary, są powierzchnie. Chociaż opisane w poprzednim podrozdziale krzywe mogą funkcjonować w układzie trójwymiarowym, to są to nadal elementy jednowymiarowe. Powierzchnie natomiast cechują się dwoma wymiarami. Są to obiekty geometryczne opisane funkcyjnie oraz parametrycznie. Opis ten odbywa się za pomocą klasycznych funkcji matematycznych, które określają ich kształt w danym układzie współrzędnych w postaci funkcji ogólnej $z = f(x, y)$. Powierzchnie w środowisku Dynamo opisane są wiel-

kościami U i V. Definiują one przestrzeń parametrów determinujących dany rodzaj i kształt powierzchni, rozszerzając tym samym liczbę parametrów definiujących powierzchnie o takie obiekty jak wektory normalne czy płaszczyzny styczne. Topologia powierzchni pokazana została na rysunku 4.29.



Rys. 4.29. Parametry definiujące powierzchnię: (1) powierzchnia, (2) krzywa izometryczna U, (3) krzywa izometryczna V, (4) współrzędna UV, (5) płaszczyzna prostopadła, (6) wektor normalny [86]

Domeną (dziedziną) powierzchni jest zakres parametrów U i V. Każdy z nich określa pojedynczy punkt (jego współrzędne trzech wymiarów) na definiowanej powierzchni. Pełna definicja domeny powierzchni to zestaw zakresów parametrów (U, V), gdzie zakres każdego z parametrów określony jest jego wartościami granicznymi, czyli (od U_{min} do U_{max}) i (od V_{min} do V_{max}). Oznaczenia te pokazano na rysunku 4.30.

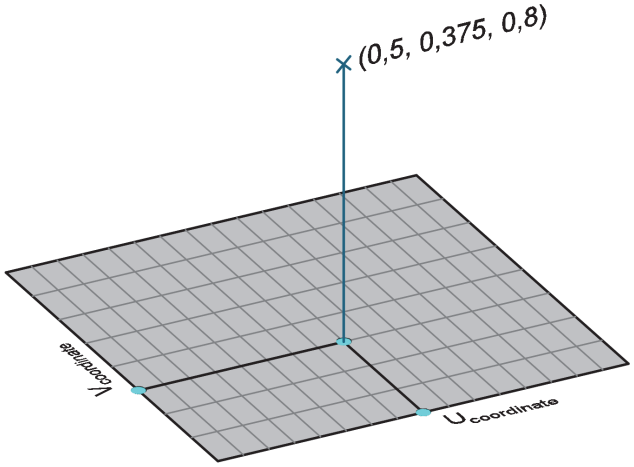


Rys. 4.30. Oznaczenia parametrów U i V definiujących powierzchnię

Domeny powierzchni określane są względnie w zakresie wartości od 0,0 do 1,0 w obu kierunkach *U* i *V*.

W przypadku gdy jeden z parametrów *U* lub *V* ma stałą wartość, a odpowiednio drugi zawiera domenę wartości, to wówczas dochodzi do zaistnienia tzw. krzywej izometrycznej lub inaczej izoparametrycznej. Przykład takich krzywych dla stałych wartości parametrów *U* i *V* pokazano na rysunku 4.29, gdzie oznaczono je jako 2 i 3. Mianem płaszczyzny prostopadłej określa się płaszczyznę, która jest tak zorientowana do obu krzywych izometrycznych *U* i *V* w punkcie o współrzędnej *UV*. Natomiast wektor, który jest prostopadły do tej płaszczyzny, określany jest jako wektor normalny.

Punkt jest definiowany przez parametry *U*, *V*, zawarte w domenie *UV*, a czasami również dodatkowo przez współrzędną *W* (rys. 4.31).

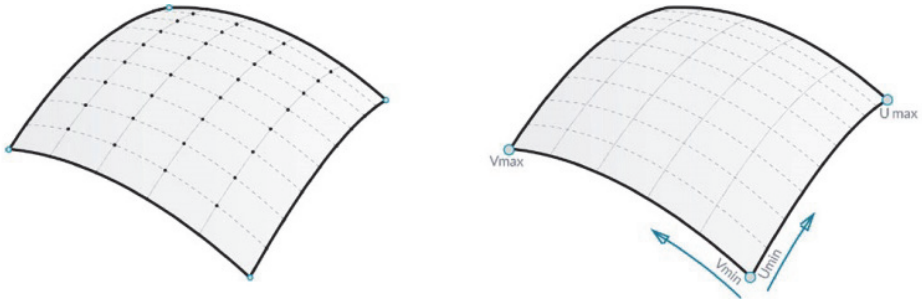


Rys. 4.31. Oznaczenia punktu na powierzchni

4.3.11. Powierzchnie NURBS

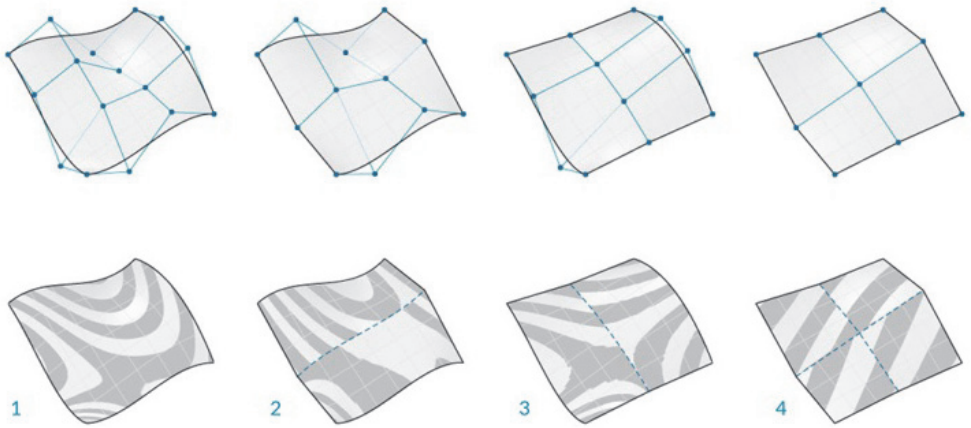
Rozwinięciem w przestrzeni dwuwymiarowej krzywych NURBS są powierzchnie NURBS. Są one tak naprawdę zbiorem wielu krzywych NURBS, które funkcjonują w dwóch kierunkach. W konsekwencji powierzchnie NURBS cechują się takimi samymi elementami jak krzywe, z których niejako się składają. Tym samym kształt tego typu powierzchni determinują punkty kontrolne, których jest o wiele więcej niż w przypadku krzywych NURBS, oraz stopień powierzchni w kierunkach *U* i *V*. Analogiczne są również algorytmy opisujące te powierzchnie, omówione szczegółowo w poprzednim podrozdziale. Powierzchnie NURBS są więc opisane i zbudowane z elementów takich jak krzywe izometryczne *U* i *V*, współrzędne *UV*, płaszczyzny prostopadłe i wektory normalne. Parametrami służącymi do opisu i sterowania tymi elementami są punkty kontrolne, wagi i stopnie powierzchni.

Geometria powierzchni jest ściśle związana z dwoma kierunkami *U* i *V*, pokazanymi na rysunku 4.32.



Rys. 4.32. Geometria powierzchni NURBS [86]

Jak już zaznaczono, powierzchnie NURBS są elementami dwuwymiarowymi, które *de facto* są zbudowane z siatki punktów kontrolnych. Użytkownik może być tu nieco zmylony, gdyż widzi powierzchnię pofalowaną i ciągłą, co należy mieć na uwadze. Wpływ stopnia powierzchni NURBS na jej kształt pokazano przykładowo na rysunku 4.33.



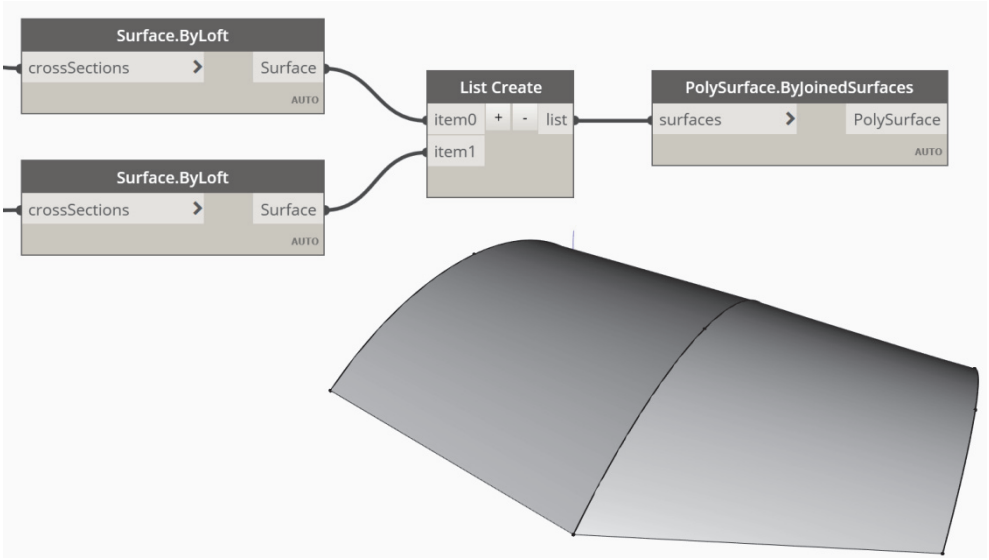
Rys. 4.33. Powierzchnie NURBS o różnych stopniach: (1) stopień (*U*, *V*) = (3, 3); (2) stopień (*U*, *V*) = (3, 1); (3) stopień (*U*, *V*) = (1, 2); (4) stopień (*U*, *V*) = (1, 1) [86]

Jak widać, im wyższy stopień, tym bardziej gładka jest powierzchnia, z uwagi na większe zagęszczenie punktów kontrolnych.

4.3.12. Polipowierzchnie

Ciekawą opcją są tzw. polipowierzchnie, które stanowią obiekt hybrydowy. Składają się one z powierzchni różnego rodzaju, które są połączone ze sobą krawędziami.

Jest to wygodne w przypadku konieczności modelowania powierzchni, które składają się fragmentami z powierzchni płaskich, a fragmentami z powierzchni zakrzywionych. Zastosowanie jednego uniwersalnego typu, np. powierzchni NURBS, byłoby w takim przypadku mniej wygodne z uwagi na konieczność sterowania punktami kontrolnymi. Przykład polipowierzchni stworzonej jako jeden obiekt geometryczny z dwóch powierzchni pokazano na rysunku 4.34.



Rys. 4.34. Scalenie dwóch powierzchni w polipowierzchnię

4.3.13. Bryły

Grupą obiektów geometrycznych najwyższego rzędu są bryły o przestrzennej, trójwymiarowej formie. Omówione zostaną one dla porządku, choć w projektowaniu inżynierskim jak dotąd są używane raczej rzadko.

Przestrzenność w ujęciu pełnej trójwymiarowości jest podstawową różnicą w odniesieniu do wszystkich opisanych uprzednio obiektów geometrycznych funkcjonujących w środowisku Dynamo. Jednakże bryły charakteryzują się jeszcze innymi cechami. Przede wszystkim dotyczy to opisu topologicznego, gdzie wyróżnia się składowe brył takie jak powierzchnie, krawędzie i wierzchołki oraz możliwość przeprowadzania operacji logicznych.

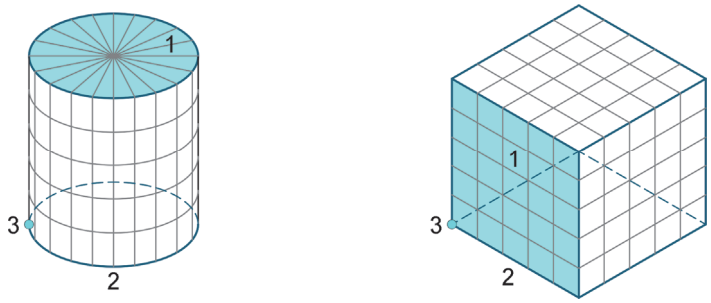
Obiekt typu bryła definiowany jest w środowisku Dynamo jako element obejmujący objętość ograniczoną przez zamkniętą obwiednię. Definiuje ona kierunek do wewnątrz i na zewnątrz bryły. Składa się z jednej lub wielu powierzchni, tworzących niejako „obudowę". Warunkiem klasyfikowania bryły jako takiej jest jej „szczelność". Bryły tworzą więc połączone ze sobą i zasklepione powierzchnie lub polipowierzchnie dowolnego w zasadzie kształtu. Bryła może również powstać w wyniku operacji logicznych przeprowadzanych na obiektach będących bryłami,

ale również w wyniku przeprowadzenia takich działań na obiektach, które bryłami nie są. Przykładowo łącząc ze sobą kilka powierzchni, można doprowadzić do tego, że powstanie obiekt zamknięty, który będzie już bryłą.

Najbardziej elementarne bryły to sześcian, prostopadłościan, walec, sfera i stożek. Przykłady niektórych obiektów tego typu (2)-(5) pokazano na rysunku 4.35 wraz z przykładem powierzchni, która bryłą nie jest (1).

Rys. 4.35. Przykłady obiektów, które są lub nie są bryłami: (1) płaszczyzna, (2) sfera, (3) stożek, (4) walec, (5) sześcian [86]

Jak już wspomniano, bryła składa się z kilku elementów składowych, tj. ścian, krawędzi i wierzchołków. Zaczynając od ostatniego z nich, wierzchołkiem określa się punkty początkowe i końcowe krawędzi, które z kolei są krzywymi definiującymi styki poszczególnych ścian. Te z kolei zbudowane są z powierzchni i to one tworzą przestrzeń i objętość brył (rys. 4.36).



Rys. 4.36. Elementy składowe brył: (1) powierzchnie, (2) krawędzie, (3) wierzchołki

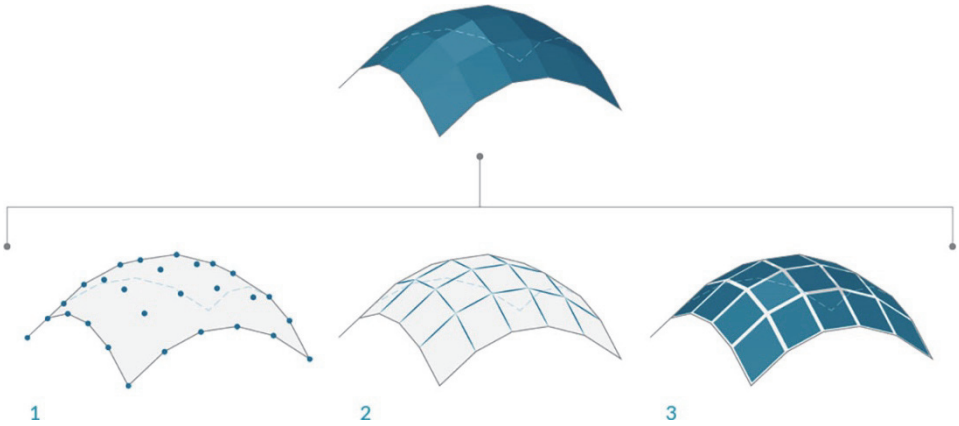
Bryły mogą być poddawane rozmaitym operacjom. Wyróżnia się operacje logiczne, które prowadzą do ich łączenia i obejmują następujące operacje elementarne: przecięcie, podzielenie, usunięcie i połączenie. Innymi operacjami są manipulacje prowadzące do wygładzania ich krawędzi, takie jak ich fazowanie i zaokrąglanie.

4.3.14. Siatki

Ostatnią grupą obiektów geometrycznych, które wykorzystywane są w modelowaniu, są siatki. Dotyczy to przede wszystkim modelowania obliczeniowego, czyli obliczeń numerycznych. Siatki reprezentują całą gamę fizycznych obiektów 3D,

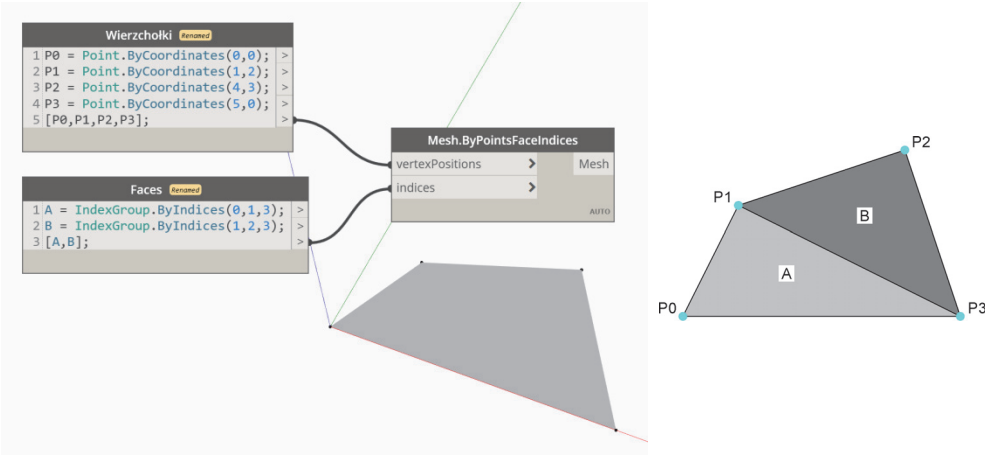
które podlegają modelowaniu i obliczeniom inżynierskim. Zastosowanie siatek wykracza daleko poza symulacje inżynierskie i obejmuje głównie modelowanie 3D.

Siatka to nic innego jak zbiór elementów trójbocznych i czworobocznych, czyli po prostu trójkątów i prostokątów. Topologia siatki wyróżnia w nich składowe identyczne jak w przypadku brył, tj. powierzchnie, krawędzie i wierzchołki oraz obiekty dodatkowe, czyli tzw. normalne (rys. 4.37).



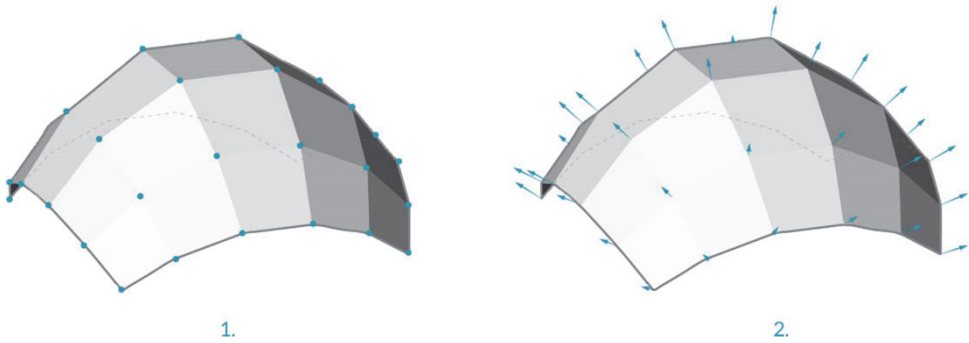
Rys. 4.37. Dekompozycja siatki [86]

W środowisku Dynamo siatki są definiowane za pomocą wierzchołków i powierzchni. Wierzchołki to nic innego jak punkty określające i rozgraniczające składowe powierzchni siatki, czyli elementarne powierzchnie trójkątne lub prostokątne. Tym samym do zbudowania siatki potrzebne są dane w postaci listy wierzchołków i algorytm ich grupowania w powierzchnie określany mianem grupy indeksów (rys. 4.38).



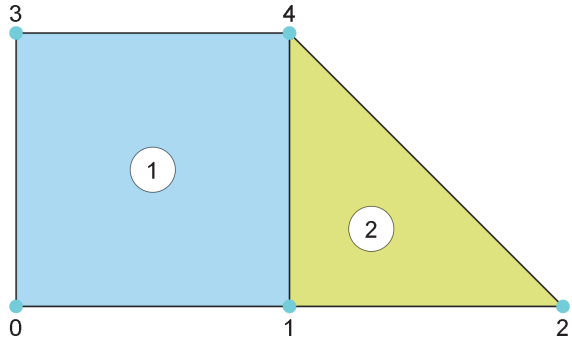
Rys. 4.38. Definicja siatki: (1) lista wierzchołków, (2) lista grup indeksów do zdefiniowania powierzchni

Tak jak już wspomniano, wierzchołki stanowią po prostu zestaw punktów definiujących poszczególne podobszary siatki, a w efekcie również jej krawędzie. W tej topologii istotne są tzw. normalne wierzchołków, będące wektorami opisującymi średni kierunek stykających się w wierzchołku powierzchni (rys. 4.39). Normalne wierzchołków określają orientację siatki „do wewnątrz" i „na zewnątrz".



Rys. 4.39. Definicja siatki: (1) wierzchołki, (2) normalne wierzchołków [86]

Powierzchnie w siatkach są określane i niejako ograniczane przez wierzchołki. Poszczególne powierzchnie stanowią więc listy trzech lub czterech wierzchołków, uszeregowane w określonym porządku. Kluczowy jest tutaj indeks wierzchołków, który porządkuje strukturę siatki, określając wzajemne relacje pomiędzy powierzchniami i wierzchołkami. Orientacja wierzchołków dla siatki złożonej z dwóch elementarnych powierzchni kwadratowej i trójkątnej pokazana została na rysunku 4.40.



Rys. 4.40. Definicja powierzchni: (1) powierzchnia czworokątna określona wierzchołkami 0-1-4-3, (2) powierzchnia trójkątna określona wierzchołkami 1-2-4

Jak widać, wierzchołki oznaczone numerami 1 i 4 są wspólne dla powierzchni 1 i 2. Tym samym przyjęta koncepcja budowy siatki oparta na indeksie wierzchołków pozwala na zmniejszenie ich liczby niezbędnej do definicji całej siatki.

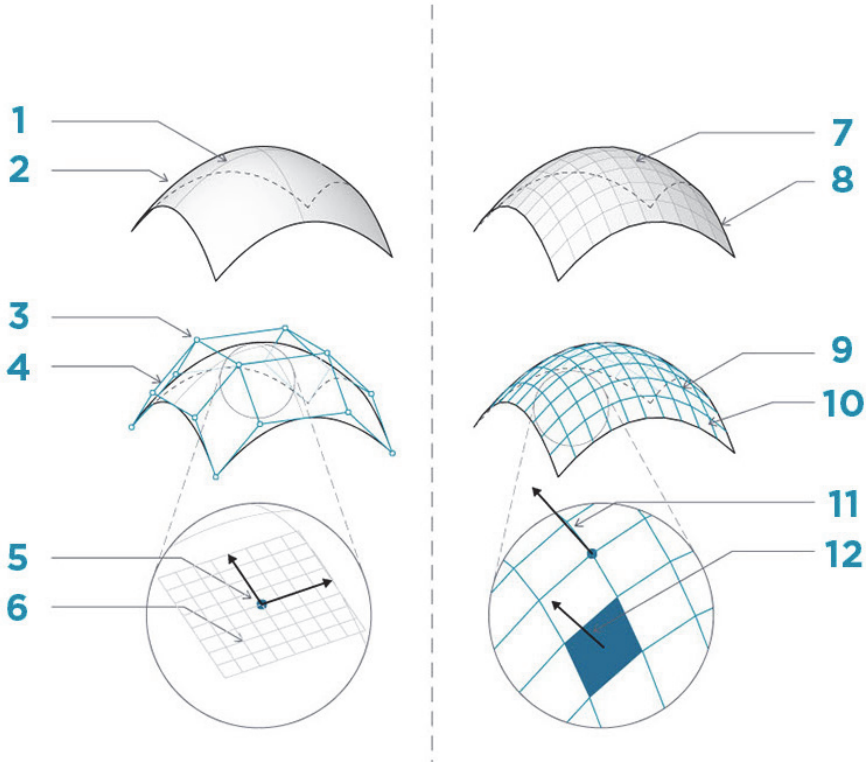
4.3.15. Siatki a powierzchnie NURBS

Na koniec należy jeszcze poświęcić kilka słów kwestiom różnic, jakie istnieją pomiędzy siatkami a geometriami typu NURBS.

Siatki są definiowane wierzchołkami i rozpiętymi pomiędzy nimi powierzchniami. Z uwagi na brak opisu matematycznego nie ma możliwości sterowania ich geometrią poprzez zmianę parametrów. Zagęszczenie siatki jest możliwe przez podwyższenie stopnia jej dokładności, pociągające zwiększenie liczby powierzchni.

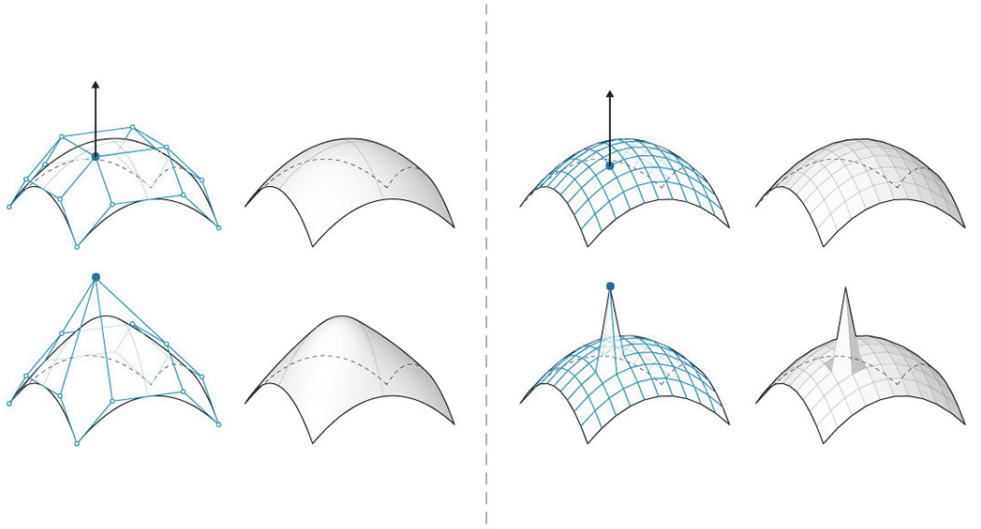
W przypadku powierzchni NURBS istnieje możliwość zmiany geometrii w oparciu o parametry sterujące. Powierzchnie te są złożone z serii krzywych NURBS, które biegną w dwóch kierunkach *U* i *V*. Parametryzacja powierzchni NURBS oparta jest na definicji domeny *UV*. Dzięki temu że krzywe składowe są opisane równaniami matematycznymi możliwe jest dokładne dostosowanie ich geometrii w oparciu o zmianę parametrów sterujących.

Porównanie siatek i powierzchni NURBS pokazano na rysunku 4.41.



Rys. 4.41. Porównanie siatek i powierzchni NURBS: (1) powierzchnia, (2) krzywa izoparametryczna, (3) punkt kontrolny powierzchni, (4) wielobok kontrolny powierzchni, (5) punkt izoparametryczny, (6) ramka powierzchni, (7) siatka, (8) naga krawędź, (9) sieć siatki, (10) krawędzie siatki, (11) normalna wierzchołka, (12) powierzchnia siatki/normalna powierzchni siatki [86]

Globalna i lokalna zmiana geometrii siatek i powierzchni NURBS to kolejna istotna kwestia decydująca o formie tych obiektów geometrycznych. Z uwagi na punkty kontrolne ich zmiana w powierzchniach NURBS powoduje daleko bardziej znaczące zmiany geometrii całego obiektu niż w przypadku zmiany pojedynczego wierzchołka siatki. Jest to związane z matematycznym opisem powierzchni NURBS, gdzie punkty sterujące określają postać opisujących je funkcji. Zmiana punktu sterującego powoduje zatem zmianę postaci funkcji, która swym zasięgiem może obejmować znaczną część powierzchni NURBS. Zależy to w dużym stopniu od stopnia powierzchni NURBS. W przypadku siatki zmiana wierzchołka powoduje niejako lokalną zmianę geometrii powierzchni, które są przyległe do niego. Analogicznie zmiany te mogą mieć oczywiście większy lub mniejszy efekt w zależności od stopnia siatki. Pokazano to schematyczne na rysunku 4.42.



Rys. 4.42. Porównanie efektu przesunięcia węzła kontrolnego powierzchni NURBS (po lewej) i siatki (po prawej) [86]

5 PRZYKŁADY KSZTAŁTOWANIA STRUKTUR INŻYNIERSKICH W ŚRODOWISKU DYNAMO

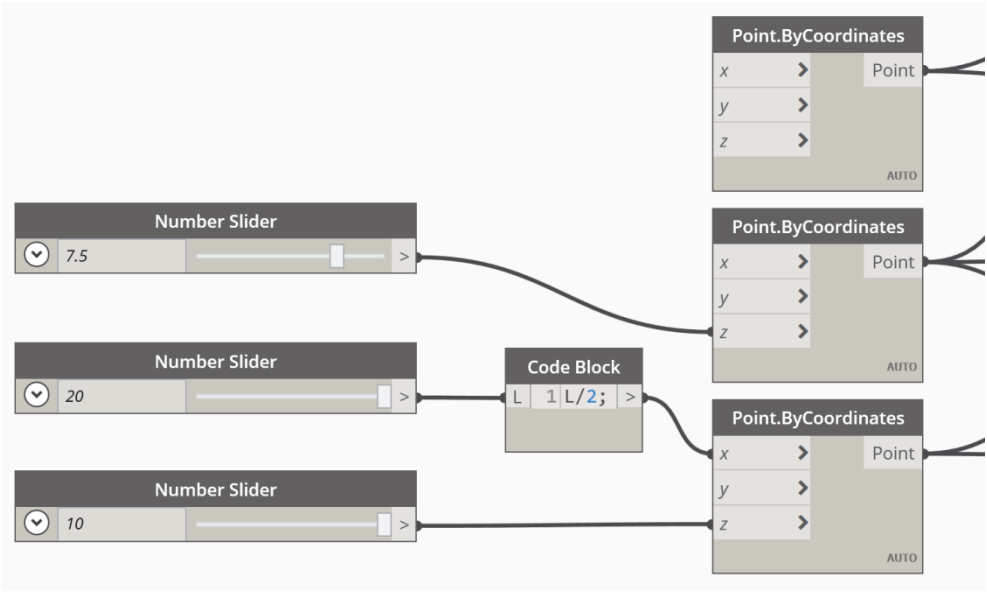
W niniejszym rozdziale przedstawiono przeglądowe skrypty, za pomocą których kształtowane mogą być w środowisku Dynamo różne struktury inżynierskie. Zostały one usystematyzowane, zaczynając od najprostszych, elementarnych przykładów programów wizualnych, gdzie głównym celem było pokazanie i niejako nauczenie czytelnika zasad kształtowania geometrii prostych elementów i układów konstrukcji budowlanych. W kolejnych podrozdziałach pokazano możliwości parametrycznego modelowania (kodowania) geometrii bardziej skomplikowanych konstrukcji. Dopełnieniem są przykłady współpracy programu Dynamo z innymi środowiskami – modelowania w Revicie i prowadzenia obliczeń w Robocie. Na koniec zaprezentowano przykład zastosowania programowania wizualnego w projektowaniu generatywnym.

5.1. RAMA PŁASKA

Najprostszym typem konstrukcji budowlanej, na podstawie którego można zaprezentować ogólną koncepcję modelowania geometrii w środowisku Dynamo, jest rama płaska. To wciąż jeden z podstawowych i najpopularniejszych układów konstrukcyjnych stosowanych choćby w budownictwie przemysłowym do kształtowania obiektów halowych. Strukturę nośną stanowią tu poprzeczne układy ram płaskich, które złożone są ze słupów i rygli (rys. 5.1).

Rys. 5.1. Konstrukcja hali o układzie ramowym [90]

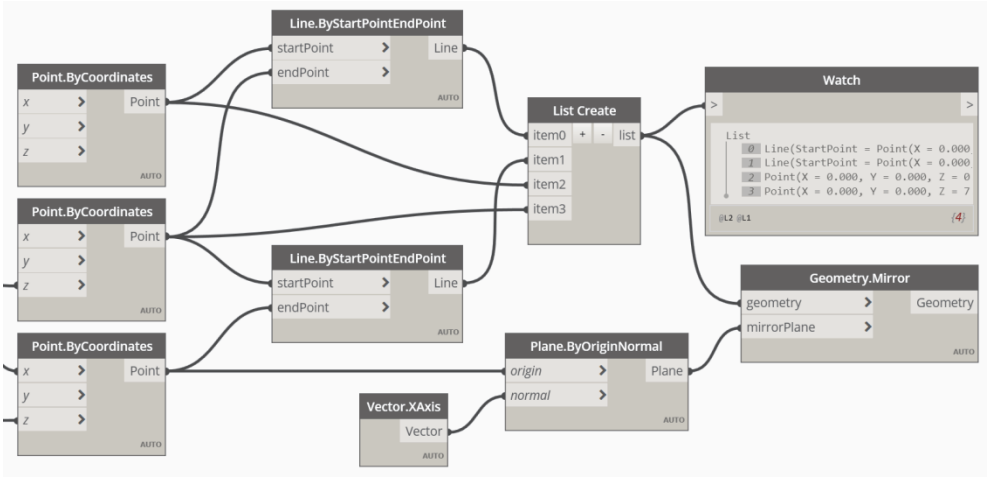
Wyjściowym zbiorem obiektów geometrycznych, na osnowie których kształtowana jest rama płaska, jest siatka punktów, które określają położenie i wymiary słupów i rygli. W skrypcie punkty te definiowane są za pomocą węzłów *Point. ByCoordinates* (rys. 5.2).



Rys. 5.2. Definicja punktów stanowiących osnowę ramy płaskiej

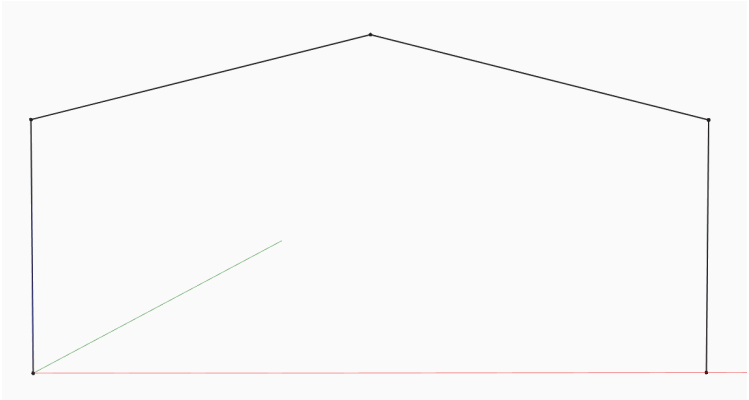
Koncepcja przyjęta w niniejszym przykładzie zakłada definicję jednej, lewej połowy ramy, która zostanie następnie wykorzystana do wygenerowania jej lustrzanego odbicia.

Jak widać na rysunku 5.2, niektóre współrzędne punktów zostały zdefiniowane za pomocą węzłów typu *Number Slider*, dzięki czemu możliwa jest ich płynna modyfikacja, pozwalająca na dynamiczne kształtowanie geometrii całego projektowanego układu konstrukcyjnego. Pomiędzy punktami rozpięte są linie tworzące słupy i rygle. Wykorzystano do tego węzły *Line.ByStartPointEndPoint*, których definicja wymaga podania na wejściu punktów początkowych i końcowych. Ostatnim krokiem jest odbicie lustrzane lewej części konstrukcji, tj. słupa, rygla i wszystkich punktów ją definiujących za pomocą węzła *Geometry.Mirror*. Do jego definicji potrzebna jest wsadowa geometria, która zostanie odbita lustrzanie, oraz określenie płaszczyzny transformacji, która została zdefiniowana za pomocą węzła *Plane.ByOriginNormal*. Należy zwrócić uwagę na grupowanie obiektów geometrycznych, które są poddawane operacji odbicia lustrzanego, które za pomocą węzła *List Create* zostały włączone do jednej listy pokazanej w węźle *Watch* (rys. 5.3).



Rys. 5.3. Definicja słupów i rygli ramy oraz operacja odbicia lustrzanego

W efekcie wygenerowana zostaje portalowa rama płaska o geometrii pokazanej na rysunku 5.4.



Rys. 5.4. Widok wygenerowanej ramy płaskiej

Opisany powyżej przykład kształtowania konstrukcji ramy płaskiej jest oczywiście bardzo elementarny, ale dobrze na nim widać ideę definiowania tego typu obiektu w skrypcie wizualnym. Poszczególne etapy to:

- definicja siatki punktów charakterystycznych,
- definicja elementów prętowych (słupy, rygle),
- operacja odbicia lustrzanego lewej części konstrukcji względem płaszczyzny symetrii.

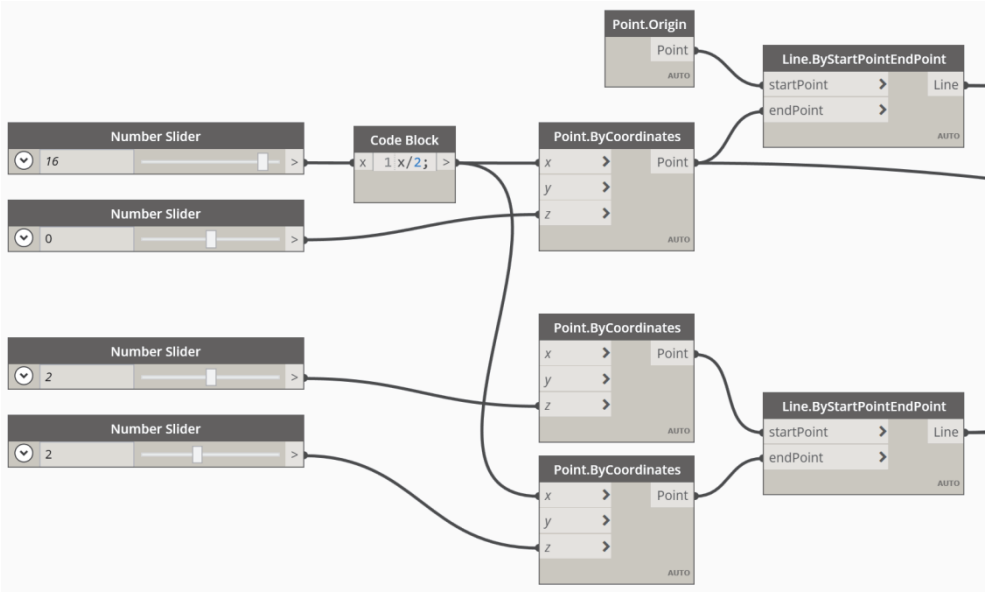
Dwa pierwsze etapy to tak naprawdę nic innego jak klasyczna definicja układów prętowych stosowana choćby w programach MES, gdzie poszczególne pręty definiowane są w oparciu o punkty początkowe i końcowe.

5.2. KRATOWNICA PŁASKA

Bardziej złożonym typem konstrukcji w porównaniu do ramy zarówno pod względem geometrycznym, jak i statycznym jest kratownica płaska. To również jeden z podstawowych i najpopularniejszych elementów konstrukcyjnych stosowanych w budownictwie. Z uwagi na efektywniejszą pracę statyczną w porównaniu do ram kratownice wykorzystywane są do kształtowania przekryć obiektów o większych rozpiętościach. Układ kratownicowy to także jeden z podstawowych typów konstrukcji mostów i wiaduktów. Standardowa kratownica składa się z pasa dolnego i górnego, krzyżulców i opcjonalnie słupków (rys. 5.5).

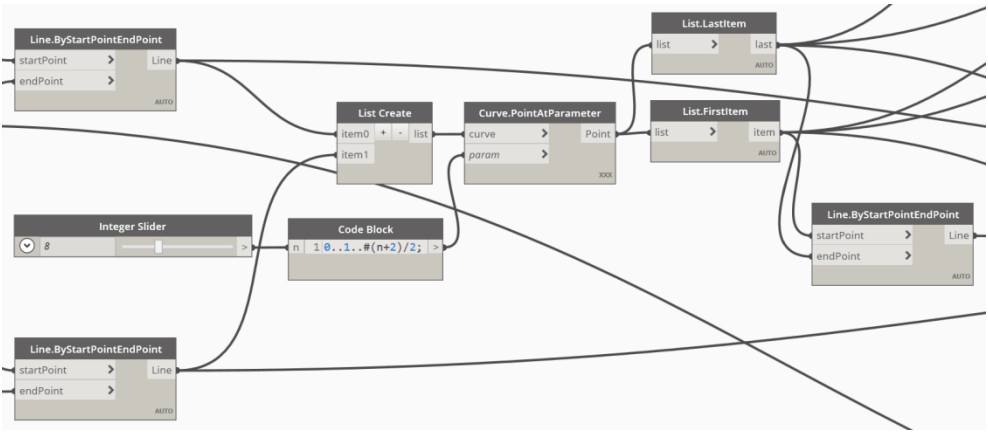
Rys. 5.5. Konstrukcja hali o układzie kratownicowym [91]

Z uwagi na większą liczbę elementów składowych oraz ich różne i alternatywne rozmieszczenie i ułożenie kształtowanie geometrii kratownicy jest bardziej skomplikowane w porównaniu do ramy płaskiej. Ogólna idea budowy skryptu wizualnego, za pomocą którego wygenerowana zostanie kratownica, zakłada podobnie jak w przypadku ramy płaskiej rozpięcie elementów składowych kratownicy na siatce punktów charakterystycznych. Tak jak poprzednio wygenerowana zostanie lewa połowa elementu, który następnie zostanie odbity lustrzanie względem płaszczyzny symetrii. W celu uzyskania elastyczności w zakresie kształtowania geometrii kratownicy, w tym także jej typów, zastosowano dynamiczną definicję punktów charakterystycznych i liczby pól za pomocą węzłów typu *Number Slider*, jak również ustawienia krzyżulców w oparciu o funkcje logiczne *True/False*. Z uwagi na specyfikę konstrukcji kratownicy w pierwszym etapie zdefiniowane zostaną punkty skrajne określające pas dolny i górny kratownicy (rys. 5.6).



Rys. 5.6. Definicja punktów charakterystycznych kratownicy

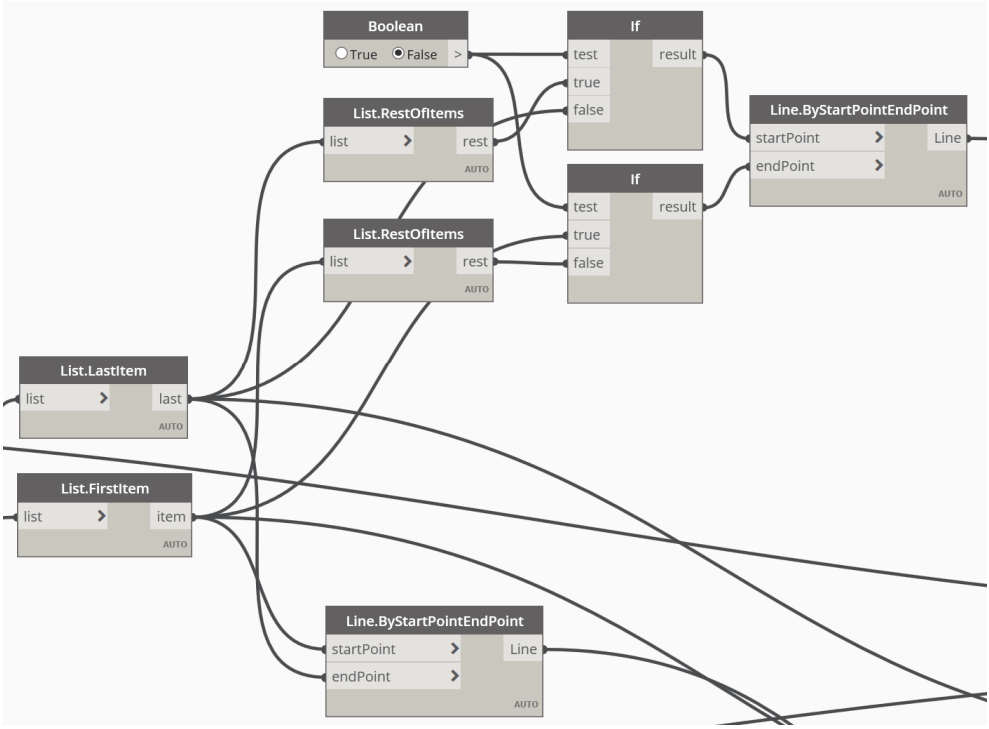
Punkty określające układ krzyżulców i słupków zostały wygenerowane za pomocą węzła *Curve.PointAtParameter*, który umożliwia przeprowadzenie takiej operacji na krzywej typu *curve* zgodnie z określonymi regułami. W niniejszym skrypcie wykorzystano w tym celu składnię definiującą punkty na pasach określające jednocześnie podział kratownicy na daną ilość pól za pomocą definicji zakresu podziału w postaci 0..1..#(n+2)/2; (rys. 5.7).



Rys. 5.7. Dyskretyzacja pasów i definicja słupków

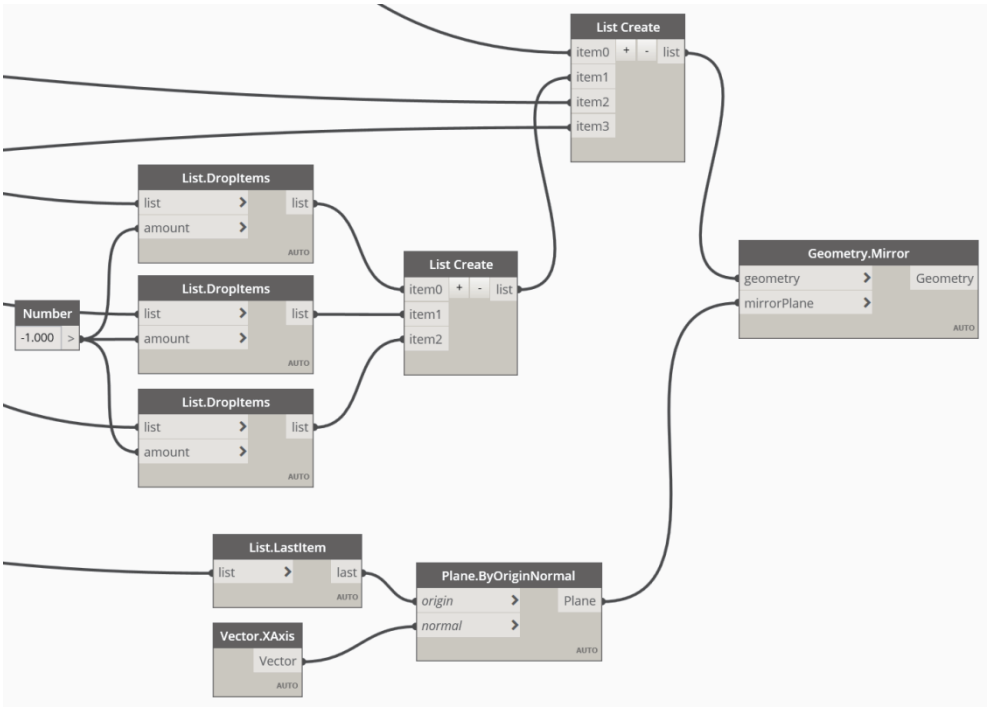
W kolejnym etapie wygenerowano słupki, które stanowią linie łączące przeciwległe punkty wygenerowane na pasach, zgodnie z procedurą opisaną powyżej.

Następnie zdefiniowano krzyżulce, które oczywiście tak jak słupki są liniami łączącymi punkty pasów, z tą różnicą że nie są to punkty przeciwległe, a przesunięte w stosunku do siebie. W tym celu przygotowano odpowiednie nowe listy punktów, wykorzystując węzły typu *List.RestOfItem* (rys. 5.8). Dodatkowo zastosowano operator logiczny *True/False*, tak aby umożliwić wybór pochylenia krzyżulców w lewą lub prawą stronę.



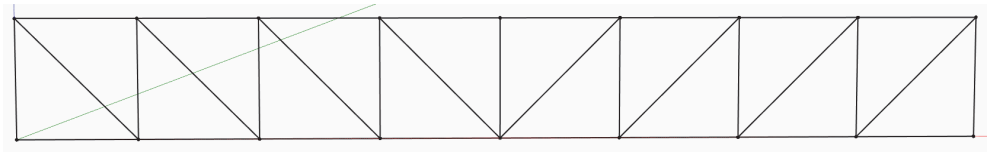
Rys. 5.8. Definicja krzyżulców

Tak przygotowaną połowę kratownicy poddano operacji lustrzanego odbicia, analogicznie jak poprzednio (rys. 5.9). Komentarza wymaga przygotowanie listy zawierającej wszystkie elementy, tj. pasy, słupki, krzyżulce i punkty. Aby nie dublować punktów i słupków leżących na osi symetrii, konieczne było ich usunięcie z grupy elementów poddanych odbiciu lustrzanemu. W tym celu zastosowano węzły pozwalające na operację na listach typu *List.DropItems*. Możliwe byłoby pozostawienie tych elementów, ponieważ z punktu widzenia dalszego wykorzystania geometrii przygotowanej kratownicy nie powodowałyby one żadnych problemów, ewentualnie konieczne byłoby ich usunięcie (w zależności od środowiska pracy).

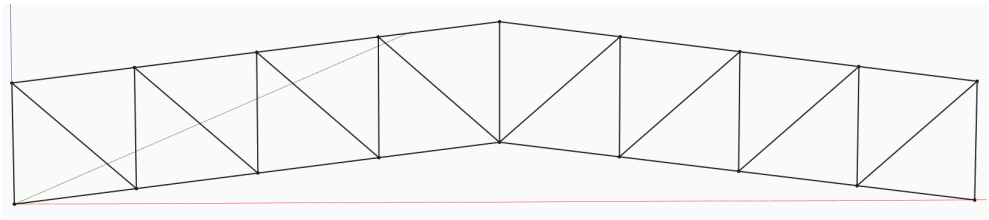


Rys. 5.9. Definicja list obiektów geometrycznych poddanych operacji odbicia lustrzanego

W efekcie możliwe było przygotowanie kratownic o praktycznie dowolnych kształtach, pokazanych przykładowo na rysunkach 5.10 i 5.11.



Rys. 5.10. Kratownica o pasach równoległych



Rys. 5.11. Kratownica o pasach pochylonych

5.3. PRZEKRYCIE QUASI-RUSZTOWE

Elementy prętowe to baza do kształtowania przestrzennych struktur nośnych, które w zależności od potrzeb lub wizji projektanta przybierają rozmaite formy. Klasycznym rozwiązaniem jest układ belkowy, gdzie elementy nośne w postaci dźwigarów lub podciągów stanowią główne elementy nośne. Na nich opierają się żebra, przyjmujące różne nazwy, w zależności od specyfiki danego układu konstrukcyjnego oraz zastosowanego materiału. Zasadniczo zaprojektowanie czy wykonanie tego typu konstrukcji nie stanowi problemu, bo najczęściej funkcjonują one w układzie płaskim, tzn. wszystkie elementy składowe znajdują się w jednej płaszczyźnie. Sytuacja ulega skomplikowaniu, gdy konstrukcja jest bardziej zmienna geometrycznie, a płaszczyzna przeradza się w powierzchnię niekoniecznie płaską. To bardzo często obserwowane rozwiązanie, które znajduje zastosowanie w wielu różnych typach obiektów budowlanych, które cechują się mniej lub bardziej swobodną formą. Przykładem tego może być zadaszenie hali Mercado de Santa Caterina w Barcelonie charakteryzujące się zakrzywionym dachem (rys. 5.12).

Rys. 5.12. Zadaszenie hali Mercado de Santa Caterina w Barcelonie [92]

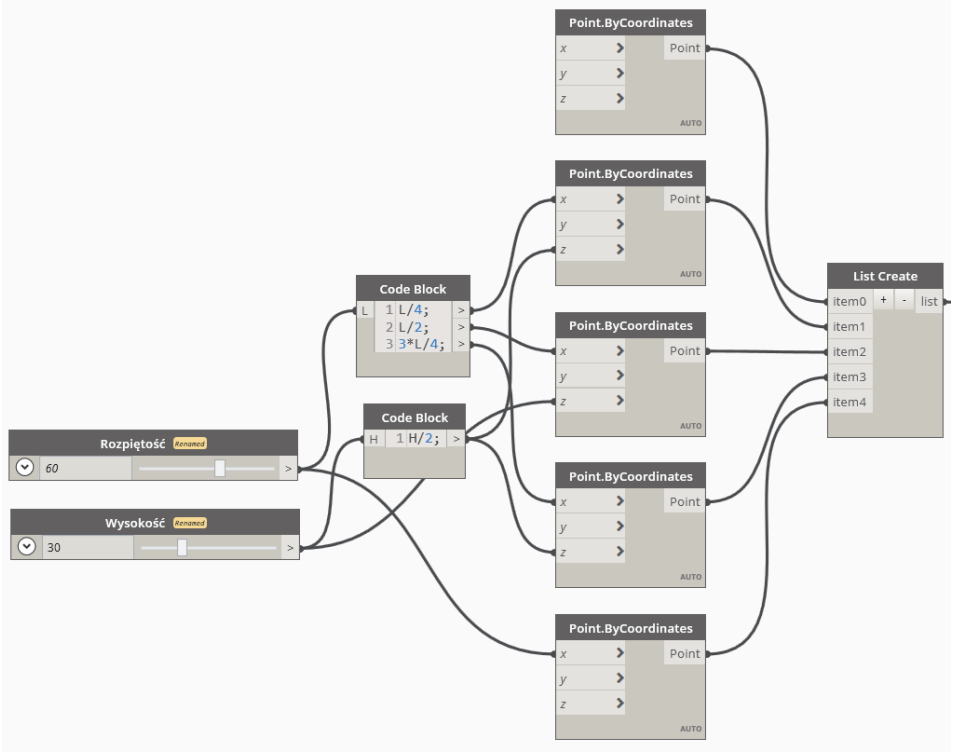
O ile sam układ konstrukcyjny nie jest tu jakoś specjalnie skomplikowany, o tyle zaprojektowanie i wykonanie samej struktury z uwagi na istniejące krzywizny nie jest już tak proste jak w przypadku choćby dachu o płaskich połaciach.

W niniejszym podrozdziale zaprezentowano przykład skryptu, za pomocą którego można kształtować tego typu konstrukcję. Oparto się w nim na układzie rusztowym, który klasycznie składa się z ortogonalnie przenikających się belek. W zależności od

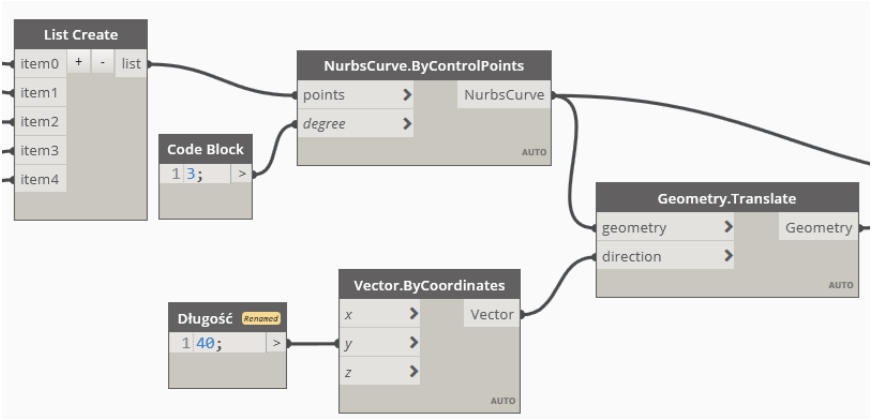
proporcji pól, poszczególne elementy rusztu przejmują taką samą lub różną część obciążenia. Zasadniczo kształtowanie i modelowanie konstrukcji rusztowych nie stanowi problemu, co podkreślono wcześniej, przywołując dwuwymiarowy belkowy układ konstrukcyjny. Sytuacja ulega diametralnej zmianie w przypadku kształtowania układów quasi-rusztowych, gdzie elementy nie stanowią jednej płaszczyzny, a jest ona definiowana powierzchnią.

Ogólna koncepcja budowy skryptu pozwalającego na zamodelowanie geometrii przekrycia w formie quasi-rusztu wpisanego w powierzchnię zakrzywioną opiera się na zdefiniowaniu jej w pierwszym kroku, a nie jak mogłoby się naturalnie zdawać dopiero po wygenerowaniu siatki elementów. W tym celu przygotowane zostaną krzywe typu NURBS niezbędne do zdefiniowania zakrzywionej powierzchni przekrycia. Analogicznie jak w poprzednich przykładach w celu elastycznej i dynamicznej modyfikacji geometrii przekrycia zastosowano węzły typu *Number Slider* (rys. 5.13). Frontową krzywą NURBS zdefiniowano za pomocą węzła *Nurbs Curve.ByControlPoints*, natomiast tylną krzywą przekopiowano z niej za pomocą opcji *Vector.ByCoordinates*. Zastosowanie krzywych NURBS pozwoliło na w miarę wierne odtworzenie przebiegu krzywych łączonych, które składają się z odcinków prostych i krzywych, co można zaobserwować w przekroju zadaszenia hali Mercado de Santa Caterina w Barcelonie (rys. 5.12).

a)



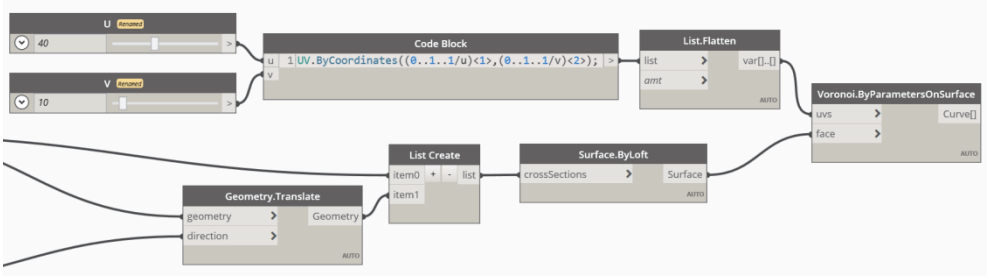
b)



Rys. 5.13. Definicja punktów i krzywych kształtujących powierzchnię przekrycia

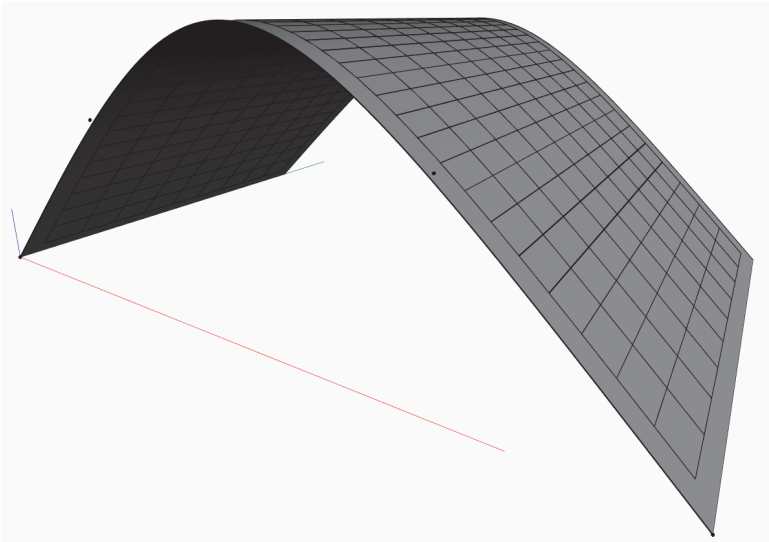
Krzywe te scalono w jednej liście, za pomocą której zdefiniowana została powierzchnia przy użyciu węzła *Surface.ByLoft* (rys. 5.13). Stanowi on bardzo wygodną opcję definiowania i kształtowania dwuwymiarowych powierzchni na bazie charakterystycznych obiektów geometrycznych niższego rzędu, takich jak np. linie.

Ostatnia część skryptu to generacja samego rusztu. W tym celu wykorzystano węzeł *Voronoi.ByParametersOnSurface*, który pozwala na stworzenie diagramu Woronoja z powierzchni na podstawie zestawu parametrów *UV*. Operacja ta jest dostępna w *Geometry.Tessellation.Voronoi.Actions*. Lista parametrów *UV* została zdefiniowana jako UV.ByCoordinates((0..1..1/u)<1>,(0..1..1/v)<2>);. Wartości parametrów podziału *U* i *V* zostały elastycznie zdefiniowane za pomocą suwaków typu *Integer Slider*. Drugim parametrem wejściowym węzła *Voronoi.ByParametersOnSurface* jest przygotowana już powierzchnia typu *Surface.ByLoft* (rys. 5.14).



Rys. 5.14. Definicja listy parametrów *UV* i diagramu Woronoja

W efekcie uzyskujemy geometrię przekrycia powierzchniowego w postaci quasi-rusztu pokazanego na rysunku 5.15.



Rys. 5.15. Model przekrycia quasi-rusztowego

Jak widać, dzięki bardzo krótkiemu i w sumie prostemu skryptowi możliwe było wygenerowanie dość skomplikowanej geometrii, jeśli wziąć pod uwagę zmienność współrzędnych wszystkich charakterystycznych punktów rusztu oraz krzywiznę samej powierzchni zadaszenia. Zmieniając proporcje pól składowych diagramu Woronoja, możliwe jest przygotowanie układu konstrukcyjnego dźwigarowo-żebrowego, co pozwoliłoby na zamodelowanie geometrii przekrycia hali Mercado de Santa Caterina w Barcelonie pokazanego na rysunku 5.12.

5.4. PRZEKRYCIE STRUKTURALNE

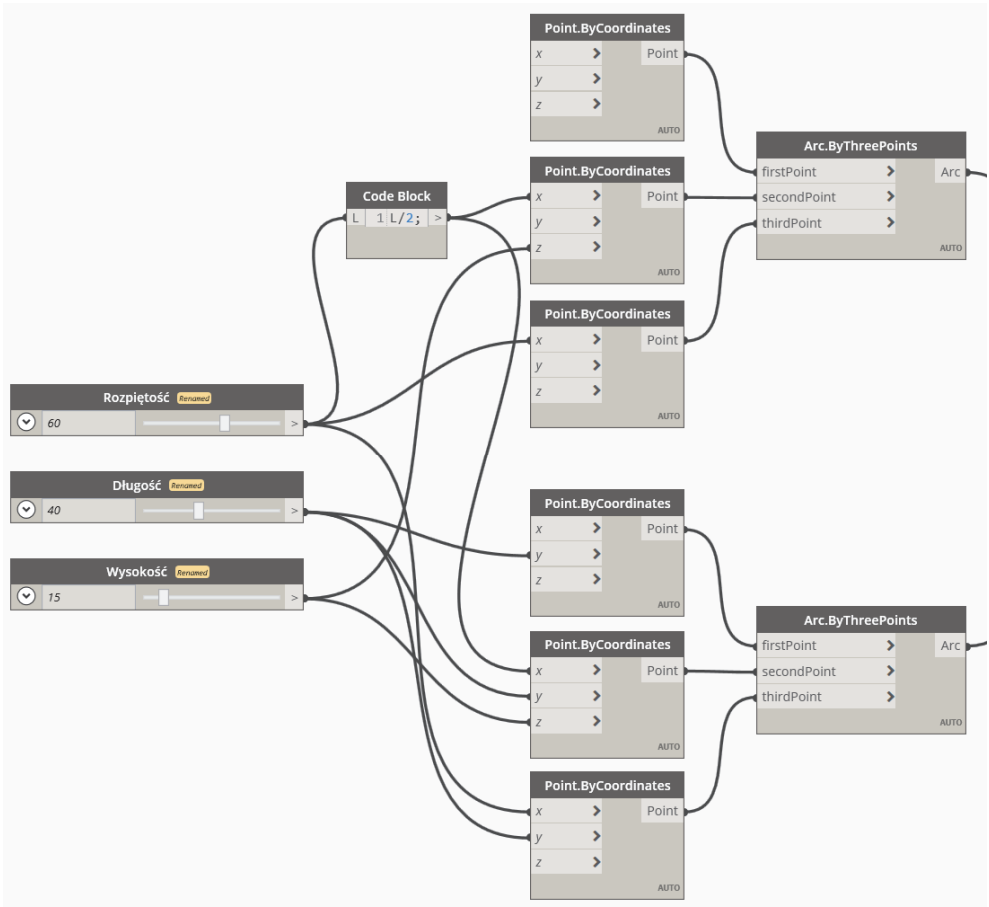
Elementy konstrukcyjne modelowane w poprzednich podrozdziałach charakteryzowały się tym, że ich składowe zorientowane były w jednej płaszczyźnie (rama, kratownica) lub jednej powierzchni (układ quasi-rusztowy). Natomiast rozwinięciem koncepcji kratownicy płaskiej jest kratownica przestrzenna, przyjmująca formę tzw. struktury. Znajduje ona zastosowanie w przypadku konieczności przekrycia obiektów o znacznych rozpiętościach lub ze względów architektonicznych (rys. 5.16). W kształtowaniu przekryć strukturalnych stosuje się różne rozwiązania i układy geometryczne, cechujące się specyficznym rozmieszczeniem elementów składowych pojedynczego modułu, z którego dana struktura jest złożona. Zasadniczo są to pasy i krzyżulce przestrzenne.

Rys. 5.16. Przekrycie strukturalne [93]

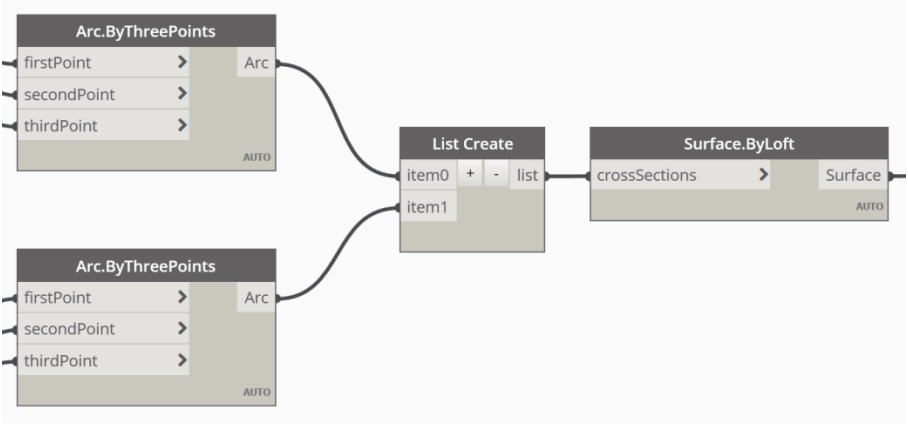
Przykład bardzo efektywnego skryptu, za pomocą którego będzie kształtowane przekrycie strukturalne o skomplikowanej geometrii przedstawiono poniżej. Na uwagę zasługuje wykorzystanie biblioteki zewnętrznej *LunchBox*, gdzie przygotowano wiele węzłów diametralnie skracających i automatyzujących pracę projektanta.

Podobnie jak poprzednio w pierwszym kroku przygotowano powierzchnię za pomocą węzła *Surface.ByLoft*, która tym razem określona została dwoma jednakowymi elementami brzegowymi – łukami (rys. 5.17).

a)

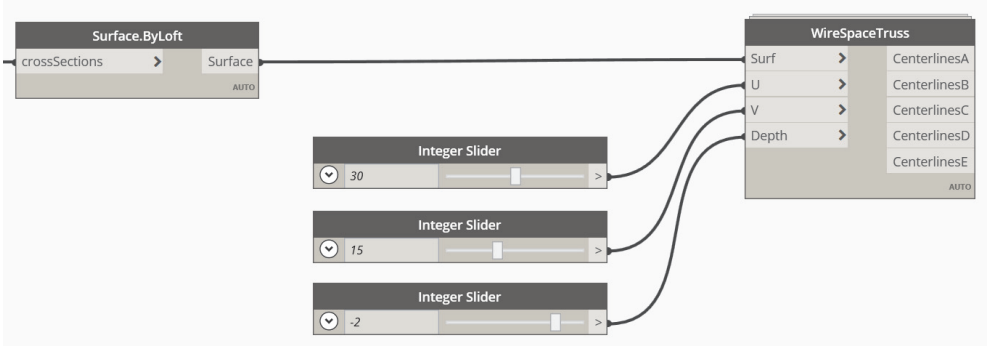


b)



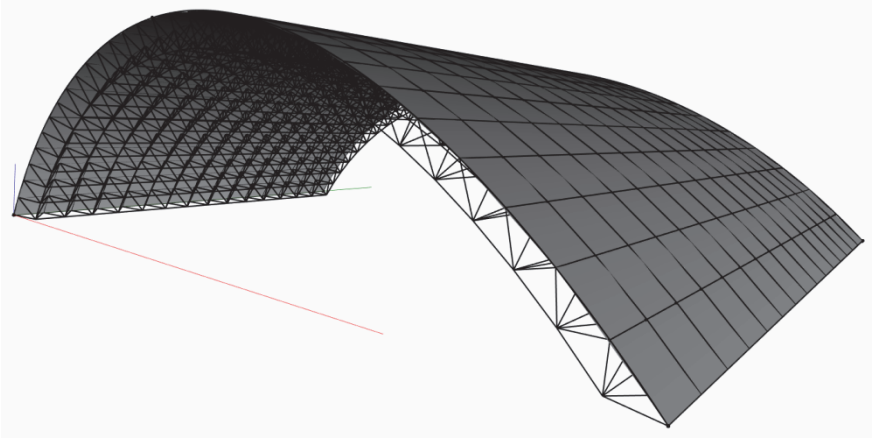
Rys. 5.17. Definicja punktów i łuków opisujących powierzchnię przekrycia strukturalnego

W celu zdefiniowania przekrycia strukturalnego wykorzystano węzeł *WireSpace Truss* (rys. 5.18), dostępny w przywołanej wyżej bibliotece *LunchBox* doinstalowanej do standardowego środowiska Dynamo. Umożliwia on automatyczną generację układu strukturalnego w postaci kratownicy przestrzennej o geometrii ostrosłupów prawidłowych o podstawie kwadratowej (piramid). Użytkownik ma możliwość sterowania podziałem przekrycia za pomocą parametrów *U* i *V*, jak również kształtowania wysokości struktury. Co równie istotne, dzięki oparciu konstrukcji strukturalnej, a dokładnie kodu węzła typu *WireSpaceTruss* na powierzchni, w bardzo elastyczny sposób można modelować przekrycia o wysokim stopniu pofalowania i wygięcia.



Rys. 5.18. Definicja geometrii przekrycia strukturalnego

Finalnie wygenerowano konstrukcję przekrycia strukturalnego pokazaną na rysunku 5.19.



Rys. 5.19. Wygenerowana konstrukcja przekrycia strukturalnego

5.5. PASAŻ ŁUKOWY

Idea projektowania parametrycznego, którą przybliżono na początku niniejszej monografii, pozwala na *stricte* matematyczny opis geometrii projektowanych obiektów. Na pierwszy rzut oka to matematyczne podejście wydaje się uciążliwe, bo wymaga od projektanta najpierw poszukania, a później dopasowania obiektu matematycznego do projektowanej struktury, tak aby uzyskać jej jak najwierniejsze odwzorowanie. Na dalszym jednak etapie związany z tym nakład pracy zwraca się z nawiązką – projektant ma o wiele szersze pole manewru w zakresie optymalizacji projektowanego obiektu. Tak naprawdę przy zastosowaniu odpowiednich algorytmów wchodzi on w zakres projektowania generatywnego, które jest kwintesencją optymalizacji projektowania. Jest to o tyle istotne, że w nowoczesnej architekturze wciąż poszukuje się nowych rozwiązań w zakresie kształtu i formy rozmaitych obiektów takich jak budynki, obiekty inżynierskie, czyli mosty, wiadukty, kładki dla pieszych, a także obiekty małej architektury o mniejszych gabarytach. Architektura poszukująca nowych kierunków i dróg rozwoju w tym zakresie wymaga nie tylko wciąż nowych koncepcji, ale też narzędzi wspomagających ten proces.

Przykładem interesującego podejścia do nowej formy projektowanych obiektów może być architektura hiszpańskiego architekta i inżyniera Santiago Calatravy, który z dużymi sukcesami od wielu już dekad kształtuje przestrzeń praktycznie całego świata. Przeważająca liczba jego prac i zrealizowanych projektów w postaci różnych obiektów inspirowana była szeroko rozumianą naturą. Przykładem mogą być obiekty wchodzące w skład kompleksu olimpijskiego w Atenach, które ukończono w roku 2004. Jednym z nich jest pasaż agory pokazany na rysunku 5.20.

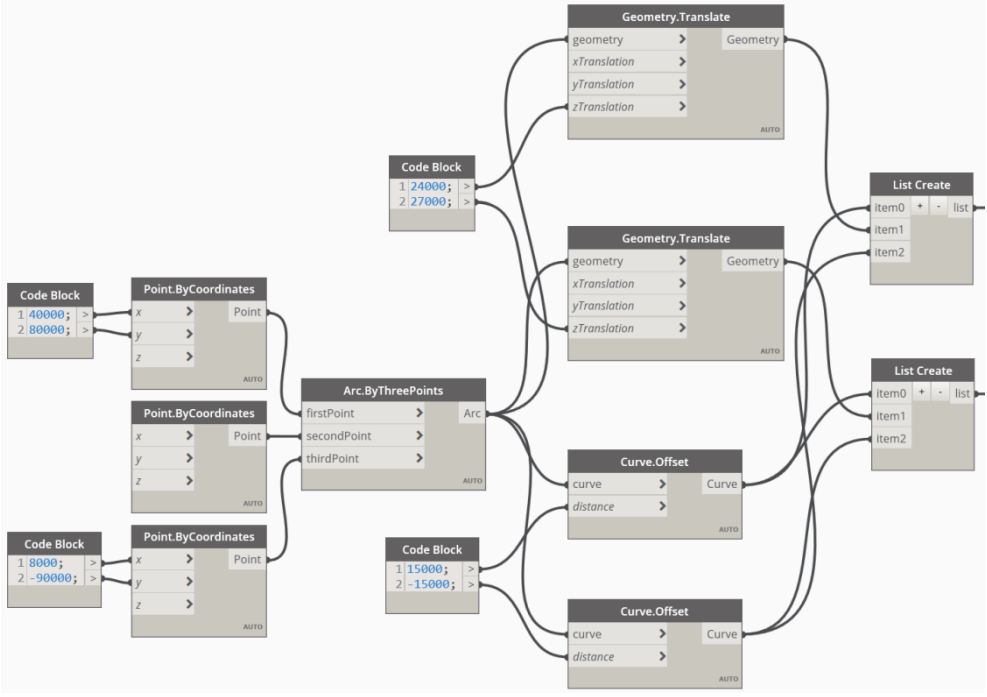
Rys. 5.20. Pasaż agory w kompleksie olimpijskim w Atenach projektu Santiago Calatravy [94]

To monumentalne ażurowe zadaszenie ograniczające od północy przestrzeń kompleksu sportowego określono właśnie mianem agory. Składa się ona z ryt-

micznie falujących łuków, zespolonych ze sobą „ścianami" składającymi się z ukośnie ułożonych prętów. Analizując geometrię tego obiektu, wyraźnie widać jej parametryczność, zarówno w rzucie, przekroju, jak i rozmieszczeniu poszczególnych elementów. W niniejszym podrozdziale zaprezentowany zostanie skrypt wizualny, za pomocą którego zamodelowana zostanie geometria zadaszenia pasażu olimpijskiej projektu Santiago Calatravy.

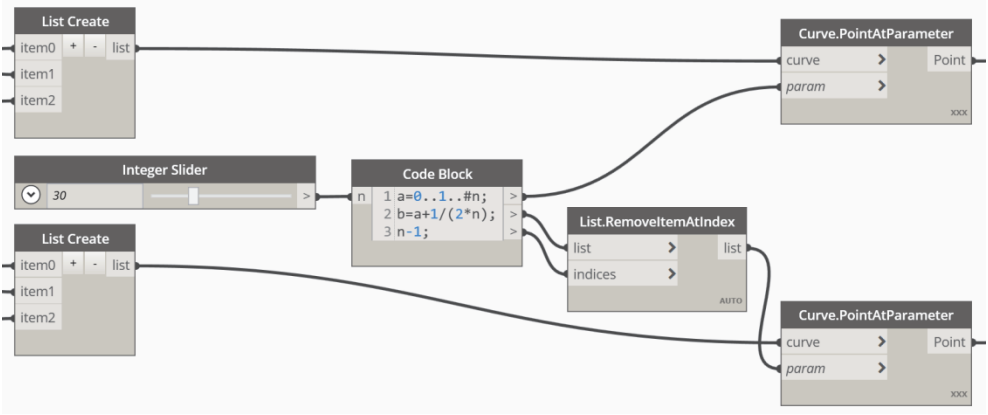
Przedstawione do tej pory elementy i struktury konstrukcyjne modelowane były za pomocą w miarę prostych i całkiem przejrzystych algorytmów, ujętych w skryptach wizualnych. O ile powierzchnie przekryć nie były płaskie, a całkiem swobodnych kształtów, o tyle sama ich generacja w oparciu o węzły *Surface.ByLoft* i elementy składowe opisujące linie typu krzywe była dość prosta. W przykładzie zaprezentowanym poniżej przedstawiona i modelowana będzie struktura o mniej zharmonizowanej geometrii charakteryzującej się zmianami rytmu. Skorzystano tu z układu skryptu zaprezentowanego w [95].

Cała koncepcja skryptu opiera się na zamodelowaniu łuków, na których wygenerowane będą punkty podziału. W pierwszym kroku zdefiniowany jest łuk podstawowy, który następnie będzie poddany operacjom kopiowania przy użyciu węzłów typu *Geometry.Translate* i *Curve.Offset* (rys. 5.21). Powstały w ten sposób układ łuków został włączony w dwie listy.



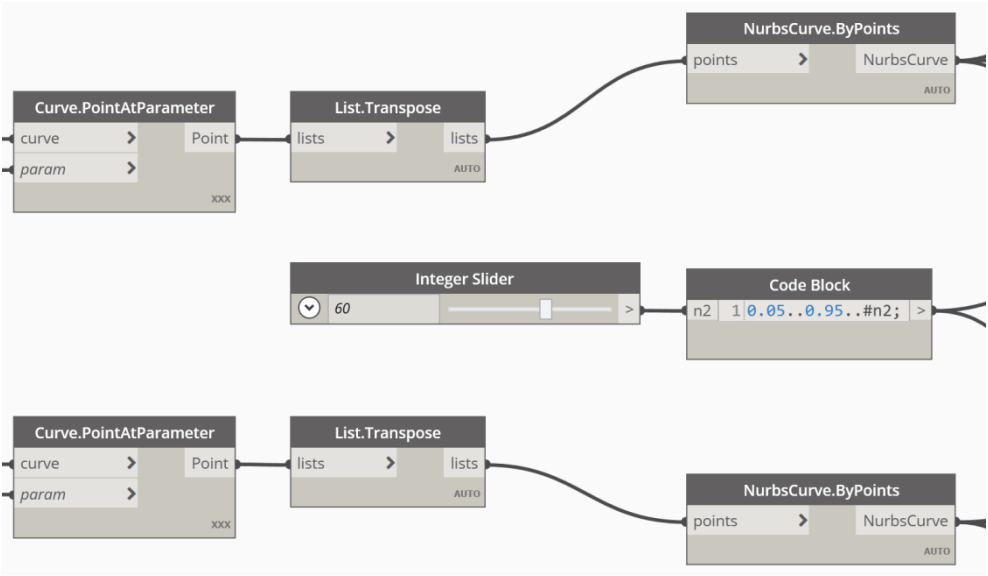
Rys. 5.21. Definicja łuków kształtujących układ przekrycia pasażu

Listy te posłużyły do generacji punktów charakterystycznych, określających rozmieszczenie, a tym samym i liczbę łuków stanowiących słupy nośne przekrycia (rys. 5.22). Poszczególne parametry przyjęto jako: a=0..1..#n;, b=a+1/(2*n);, n-1;.



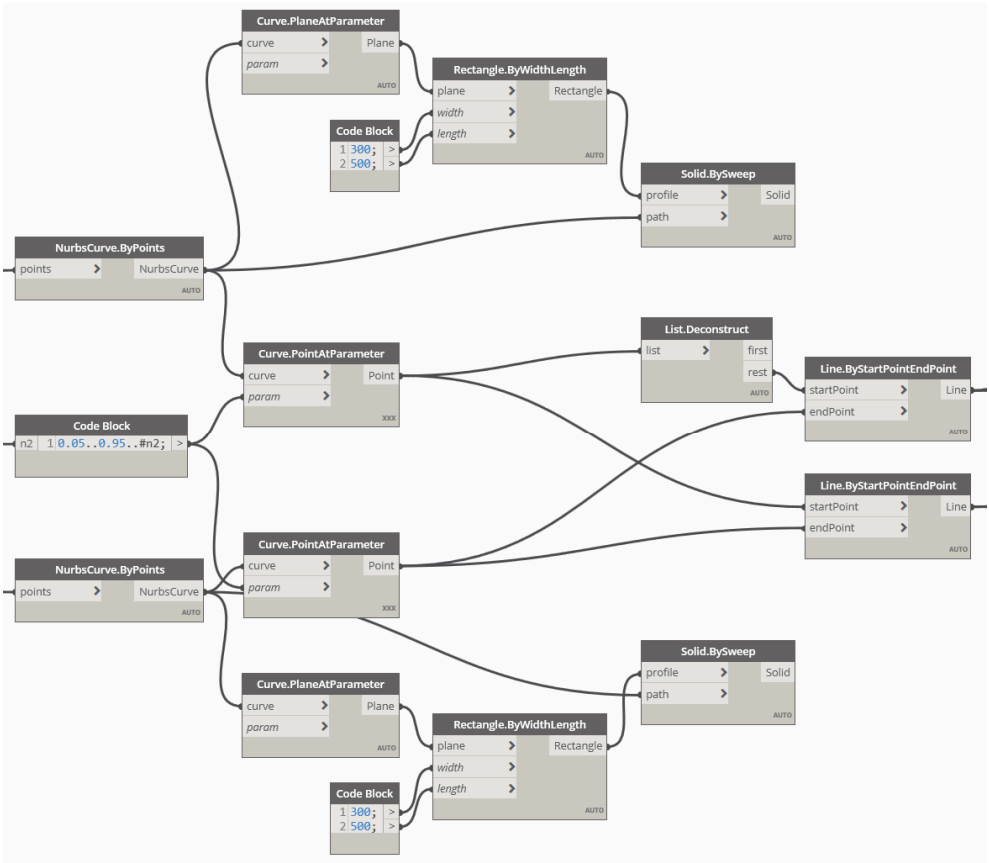
Rys. 5.22. Generacja list punktów charakterystycznych

Następny krok to definicja geometrii łuków – słupów nośnych za pomocą węzłów *NurbsCurve.ByPoints* (rys. 5.23). Zostały one utworzone jako krzywe typu BSPline poprzez interpolację pomiędzy wcześniej wygenerowanymi punktami.



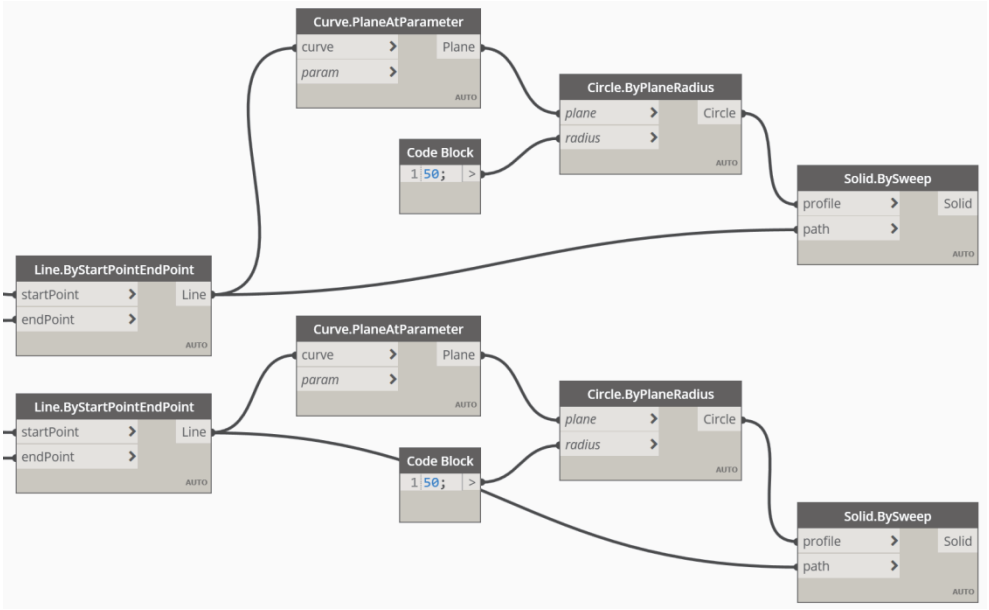
Rys. 5.23. Definicja słupów – łuków głównych

Podobną operację wykonano w celu skonstruowania elementów łączących poszczególne słupy ze sobą. Za pomocą znanej już opcji *Curve.PointAtParameter* wygenerowano punkty o określonym podziale na łukach – słupach nośnych (rys. 5.24). W tym miejscu skryptu zamodelowano słupy nośne jako elementy przestrzenne za pomocą opcji *Solid.BySweep* i określono również ich przekroje prostokątne za pomocą węzłów *Rectangle.ByWidthLength*.



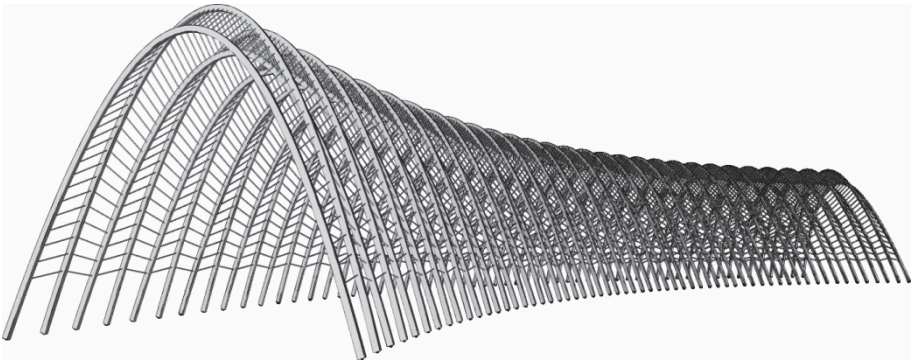
Rys. 5.24. Modelowanie łuków głównych i generacja prętów je łączących

Mając siatkę punktów, na łukach połączono sąsiednie punkty w celu wygenerowania elementów łączących słupy – łuki nośne (rys. 5.24). Analogicznie jak poprzednio wygenerowano ich przekroje, o kołowej formie przy użyciu opcji *Solid. BySweep* (rys. 5.25).



Rys. 5.25. Generacja przekrojów prętów łączących łuki główne

W efekcie uzyskano geometrię przekrycia (rys. 5.26) podobnego do zadaszenia pasażu agory w kompleksie olimpijskim w Atenach przedstawionego na rysunku 5.20.



Rys. 5.26. Konstrukcja przekrycia *à la* Calatrava

5.6. ZADASZENIE SPARAMETRYZOWANE TRYGONOMETRYCZNIE

Parametryczny opis geometrii, który do tej pory zastosowano przy okazji modelowania struktur nośnych, opierał się na opisie elementów składowych za pomocą podstawowych elementów geometrycznych, takich jak linia, łuk, koło, elipsa itd. Sterowanie ich kształtem i umiejscowieniem odbywało się poprzez zmianę parametrów opisujących je funkcji, i choć miało ono cechy matematycznej parametryzacji,

to praktycznie było realizowane w sposób niejawny. Użytkownik ręcznie bądź płynnie za pomocą węzłów *Number Slider* zmieniał najczęściej punkty charakterystyczne, powodując *de facto* zmianę funkcji opisujących wyżej wymienione elementy.

Projektowanie parametryczne daje projektantowi możliwości ograniczone jedynie aparatem matematycznym. Interesujące są przykłady obiektów, których kształt lub geometria elementów składowych opisana jest bardziej specyficznymi funkcjami w porównaniu choćby do paraboli, elipsy czy funkcji wykładniczej. Przykładem takiego obiektu może być South Pond Pavilion wybudowany w Chicago, pokazany na rysunku 5.27.

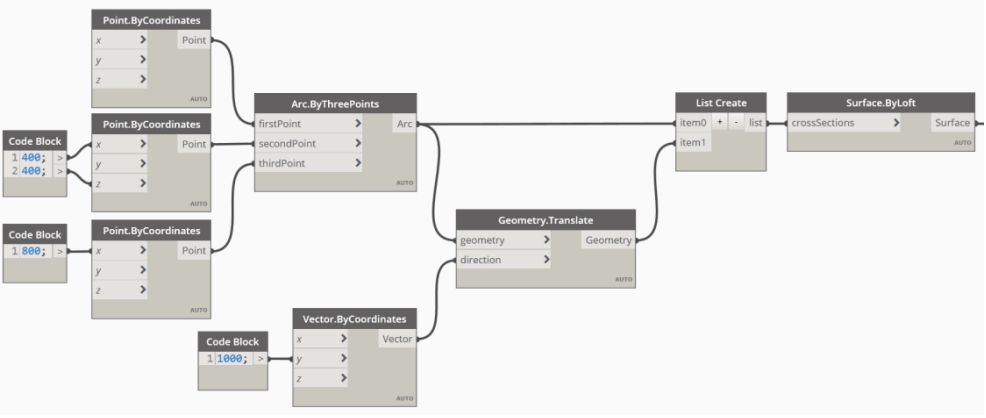
Rys. 5.27. South Pond Pavilion, Chicago [96]

Ten ciekawy obiekt na pierwszy rzut oka może kojarzyć się z wygiętym plastrem miodu ze względu na geometrię i układ „oczek" kształtujących jego bryłę i strukturę. Jednakże South Pond Pavilion skonstruowano z pofalowanych w swej płaszczyźnie łuków z drewna klejonego, zespolonych ze sobą stykającymi się ścianami. Nie wytężając zbytnio swej wyobraźni, łatwo można zauważyć, że przebieg fal poszczególnych łuków można opisać z większą lub mniejszą dokładnością za pomocą funkcji trygonometrycznych takich jak funkcje *sinus* i *cosinus*. To niejako modelowy przykład konstrukcji opisanej ściśle matematycznie!

W środowisku Dynamo, tak jak w innych tekstualnych językach programowania, użytkownik ma możliwość zdefiniowania własnego równania, opisującego geometrię kształtowanej struktury. Na niniejszym przykładzie pokazane zostanie, jak praktycznie można z tego skorzystać. Poniżej zaprezentowano skrypt umożliwiający generację przekrycia o formie organicznej, której geometria opisana jest za pomocą

funkcji trygonometrycznych. Dopasowując umiejętnie ich przebieg, możliwe będzie wygenerowanie struktury przypominającej South Pond Pavilion w Chicago.

Szkieletem konstrukcji jest obrys jej przekroju wygenerowany za pomocą łuku. Został on zdefiniowany na bazie trzech punktów charakterystycznych za pomocą węzła *Arc.ByThreePoints*. Projektowana struktura rozciągnięta jest pomiędzy dwoma krzywymi, co uzyskano, kopiując pierwszą z nich o określoną długość. Przekrycie zamknięto płaszczyzną typu *SurfaceByLoft* definiowaną za pomocą krzywych przekrojowych typu NURBS (rys. 5.28).

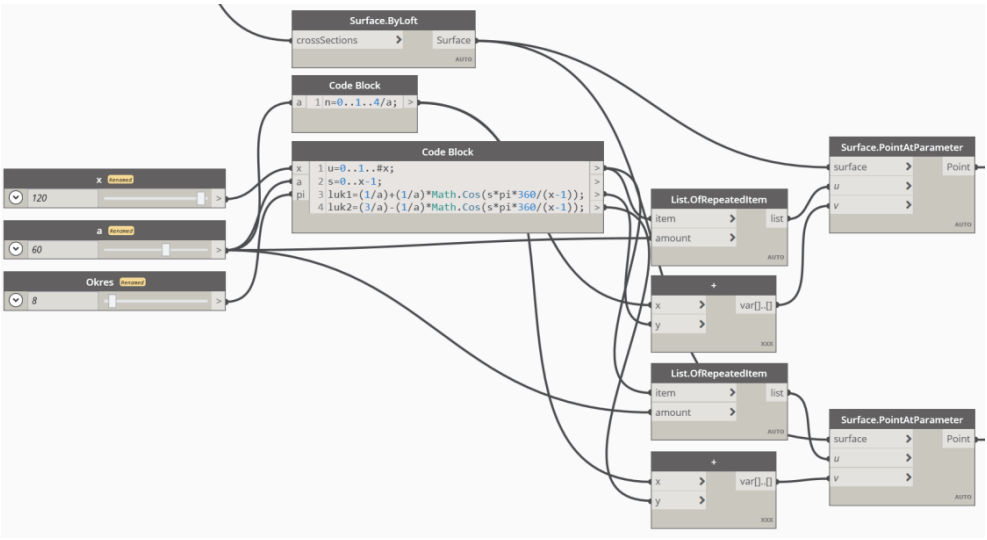


Rys. 5.28. Definicja szkieletu konstrukcji przekrycia

Druga część skryptu to definicja układu elementów wypełniających ściany przekrycia. Zastosowano następującą formułę opisującą ich geometrię:

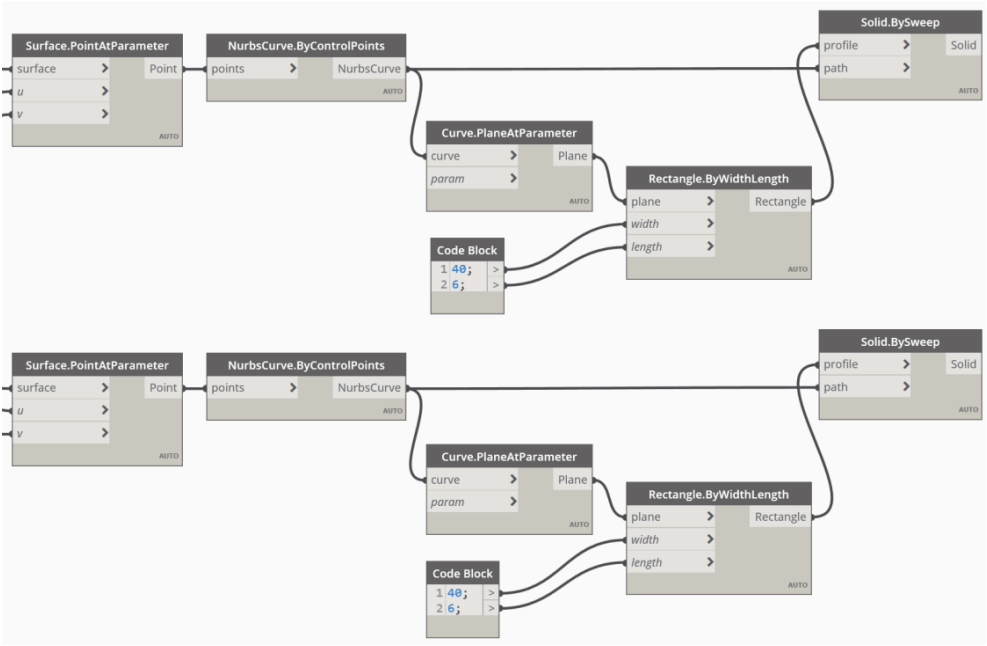
```
u=0..1..#x;
s=0..x-1;
luk1=(1/a)+(1/a)*Math.Cos(s*pi*360/(x-1));
luk2=(3/a)-(1/a)*Math.Cos(s*pi*360/(x-1));.
```

Jak widać, za pomocą funkcji trygonometrycznych można formować rzeczywiste kształty, przebiegi elementów konstrukcyjnych. Z tak zdefiniowanych krzywych pobrano punkty dla określonych parametrów *U* i *V* za pomocą węzła *Surface. PointAtParameter* (rys. 5.29).



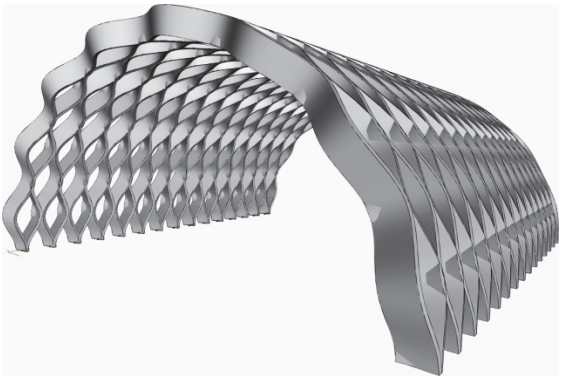
Rys. 5.29. Definicja punktów charakterystycznych określających przebieg fal łuków nośnych

W ostatnim etapie wygenerowano pomiędzy wyżej wymienionymi punktami charakterystycznymi krzywe typu *B-Spline* za pomocą węzła *NurbsCurve.ByControl Points*. Nadano im przekroje przy użyciu funkcji *Solid.BySweep* i *Rectangle.By WidthLength*, analogicznie jak to opisano w poprzednich podrozdziałach (rys. 5.30).



Rys. 5.30. Nadanie przekrojów elementom konstrukcyjnym

W efekcie wygenerowano przekrycie o kształcie analogicznym do South Pond Pavilion w Chicago (rys. 5.31).



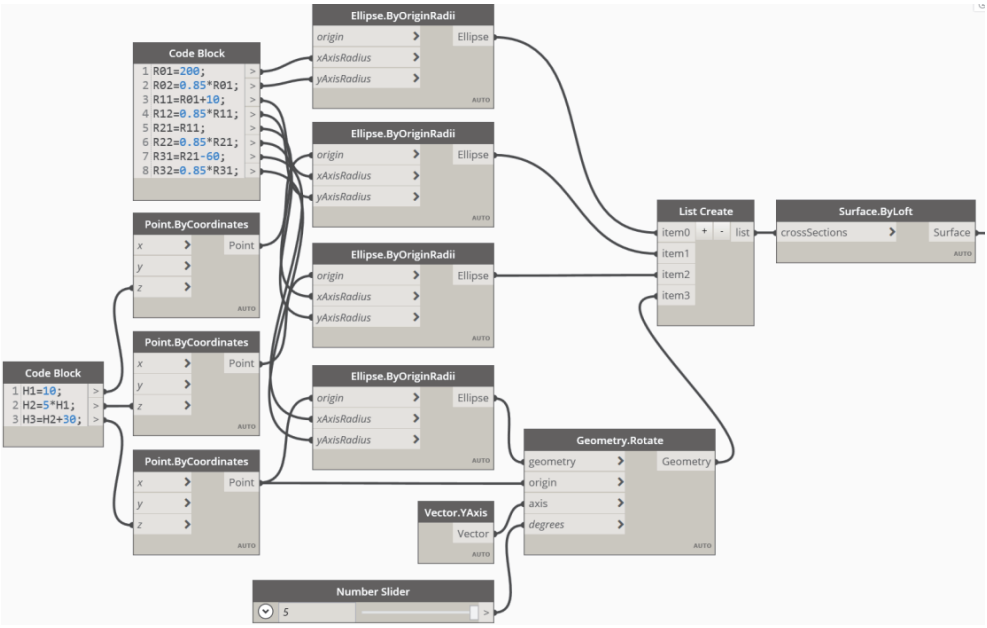
Rys. 5.31. Model South Pond Pavilion w Chicago

5.7. STADION SPORTOWY

Parametryzacja opisu geometrii jest bardzo efektywna na etapie kształtowania bryły projektowanego obiektu, a więc w fazie koncepcji, co wielokrotnie już przywoływano. Zastosowanie programowania wizualnego na tym etapie jest nad wyraz wydajne, bo daje bardzo dużą liczbę wariantów, co umożliwia optymalizację kształtowania zarówno bryły, jak i innych parametrów obiektu. Temu jest poświęcony niniejszy podrozdział, w którym pokazano możliwości w zakresie modelowania bryły i konstrukcji modelu stadionu sportowego. Modelowym przykładem w tym zakresie będzie bryła stadionu PGE Arena oddanego do użytku w roku 2011 w Gdańsku (rys. 5.32).

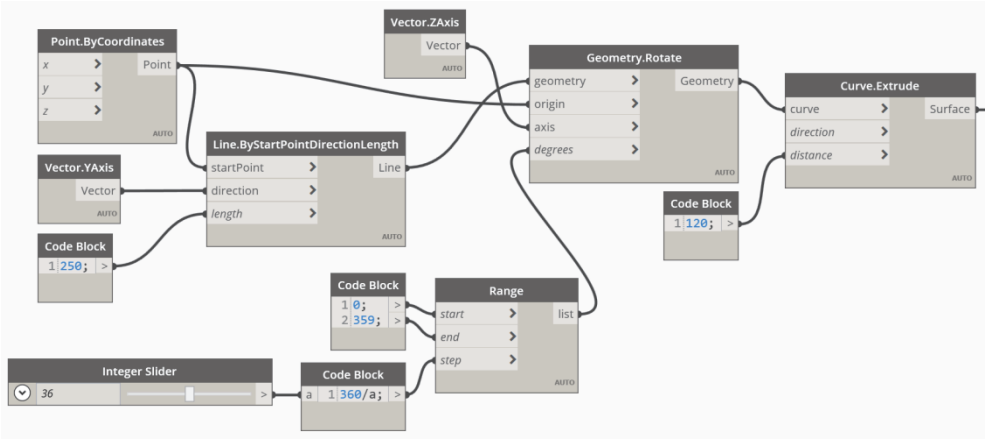
Rys. 5.32. Stadion PGE Amber Arena, Gdańsk [97]

Ogólna koncepcja stworzenia bryły projektowanego stadionu opiera się na jej opisie za pomocą elementów powierzchniowych. Są one rozpięte pomiędzy piętrowo ułożonymi obrysami modelowanymi elipsami, ale możliwe jest tu oczywiście zastosowanie elementów o innych kształtach (okręgi, krzywe łączone itp.). Do ich generacji wykorzystano węzeł *Ellipse.ByOriginRadii*, który definiuje elipsę na podstawie jej punktu środkowego oraz połówek odcinków osiowych w kierunku *x* i *y*. W prezentowanym skrypcie przyjęto cztery współśrodkowo ułożone elipsy tworzące bryłę stadionu, których poziomy zdefiniowane zostały jako współzależne, zgodnie z regułami: H1=10; H2=5*H1; H3=H2+30. Z kolei poszczególne średnice opisane zostały jako R01=200; R02=0.85*R01; R11=R01+10; R12=0.85*R11; R21=R11; R22=0.85*R21; R31=R21-60; R32=0.85*R31. Bryła stadionu została rozpięta pomiędzy elipsami składowymi za pomocą dobrze już znanej opcji *Surface.ByLoft*. Dodatkowo uwzględniono możliwość rotacji wyższych pięter bryły stadionu za pomocą węzła *Geometry.Rotate* (rys. 5.33).



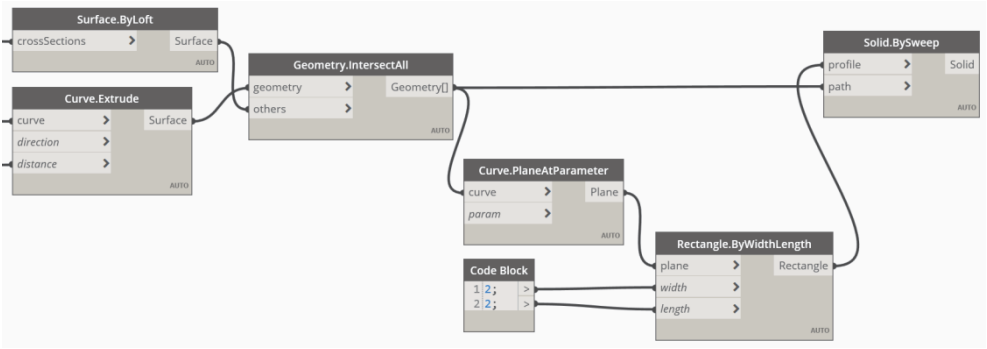
Rys. 5.33. Definicja bryły stadionu

Druga część skryptu to modelowanie konstrukcji wsporczej stadionu, którą stanowi układ łuków wpisanych w krzywiznę jego bryły. Łuki zdefiniowano jako elementy będące wynikiem operacji przecięcia płaszczyzn radialnie rozstawionych względem osi stadionu i powierzchni bryły stadionu. Wykorzystano w tym celu w kolejnych krokach węzły typu *Line.ByStartPointDirectionLength* i *Geometry.Rotate*, *Curve.Extrude*, co pozwoliło na transformację linii na płaszczyzny zorientowne radialnie w stosunku do osi stadionu. Kolejny krok to pobranie przecięcia geometrii wsadowych, tj. płaszczyzn i bryły stadionu (5.34).



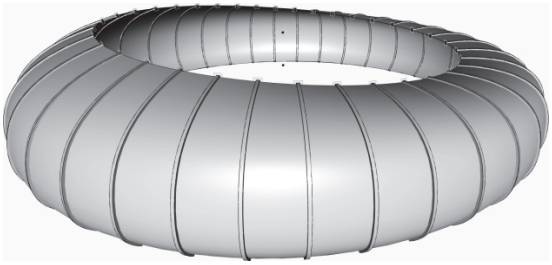
Rys. 5.34. Definicja konstrukcji stadionu

Ostatni etap to nadanie przekrojów łukom nośnym za pomocą funkcji *Solid. BySweep* i *Rectangle.ByWidthLength* (rys. 5.35).



Rys. 5.35. Nadanie przekrojów elementom konstrukcji stadionu

Przykładową bryłę stadionu pokazano na rysunku 5.36. Dzięki zastosowaniu opcji rotacji na etapie definicji elementów składowych możliwe jest uzyskanie bryły niesymetrycznej o dowolnie pochylonych ścianach.



Rys. 5.36. Bryła stadionu ze zdefiniowaną konstrukcją nośną

Na koniec należy zaznaczyć, że dzięki wykorzystaniu elips jako szkieletu bryły stadionu użytkownik w bardzo łatwy sposób może wygenerować bryłę o przekroju kołowym, bez konieczności definicji okręgów. W tym celu wystarczy zdefiniować równe promienie elipsy.

5.8. INTEGRACJA DYNAMO

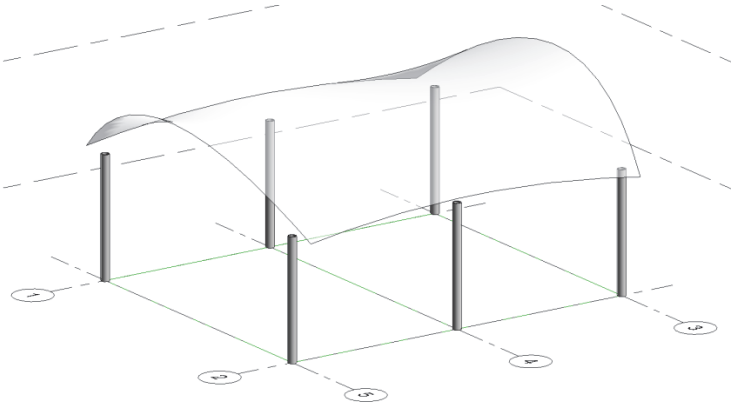
Modelowanie geometrii konstrukcji nośnych i ogólnie kształtowanie obiektów 3D to jeden z podstawowych, ale oczywiście nie jedyny, obszarów zastosowania programowania wizualnego w projektowaniu w ogóle. Jak już wspomniano, program Dynamo funkcjonuje jako samodzielne środowisko Dynamo Snadbox i Dynamo Studio, jak również komponent programu Autodesk Revit. Od roku 2021 został również włączony jako składnik programu Autodesk Robot Structural Analysis 2022. Tym samym w przywołanych programach Dynamo stało się zintegrowanym dodatkiem. W kolejnych podrozdziałach pokazane zostały przykłady współpracy oraz funkcjonalności programu Dynamo w środowisku modelowania obiektów budowlanych Autodesk Revit 2022 i prowadzenia obliczeń inżynierskich Autodesk Robot Structural Analysis 2022.

5.8.1. Współpraca z Autodesk Revit

Program Dynamo został przewidziany jako dodatek do standardowej instalacji środowiska Revit już od kilku jego wydań. Jest umiejscowiony w zakładce *Zarządzaj*, gdzie umieszczono również *Odtwarzacz Dynamo*, czyli *Dynamo Player*. Za jego pomocą użytkownik ma możliwość wyświetlania podglądów, wyboru i uruchomienia skryptów Dynamo z jednego okna dialogowego. Po uruchomieniu programu Dynamo użytkownik ma do dyspozycji analogiczne środowisko programowania wizualnego jak w przypadku samodzielnej wersji Dynamo Sandbox.

Przykład zaprezentowany poniżej pokazuje możliwości, jakie daje zastosowanie programowania wizualnego w kształtowaniu modeli obiektów budowlanych od razu w środowisku Revit, jak również obustronne wykorzystanie i transfer danych pomiędzy programami Dynamo – Revit. Wykorzystano również przykładową opcję, za pomocą której można w bardzo prosty sposób modelować konstrukcję nośną przekrycia o dowolnym kształcie, wykorzystując programowanie wizualne.

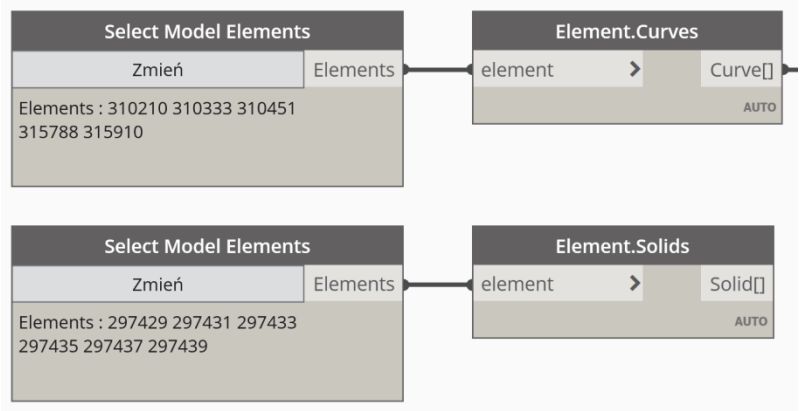
W pierwszym kroku przygotowana została, pokazana na rysunku 5.37, konstrukcja zadaszenia opartego na słupach w środowisku Revit.



Rys. 5.37. Konstrukcja zadaszenia przygotowana w środowisku Revit

Geometria zadaszenia zamodelowana została w oparciu o opcję pozwalającą na stworzenie dachu na niepionowej powierzchni bryły, wykorzystując opcję *Dach wg powierzchni*. Jest on wstępnie przewidziany do oparcia na 6 słupach. Kluczową sprawą jest wrysowanie linii pomiędzy wszystkimi słupami (rys. 5.37). Zostaną one wykorzystane w skrypcie wizualnym Dynamo do automatycznej generacji konstrukcji nośnej dachu.

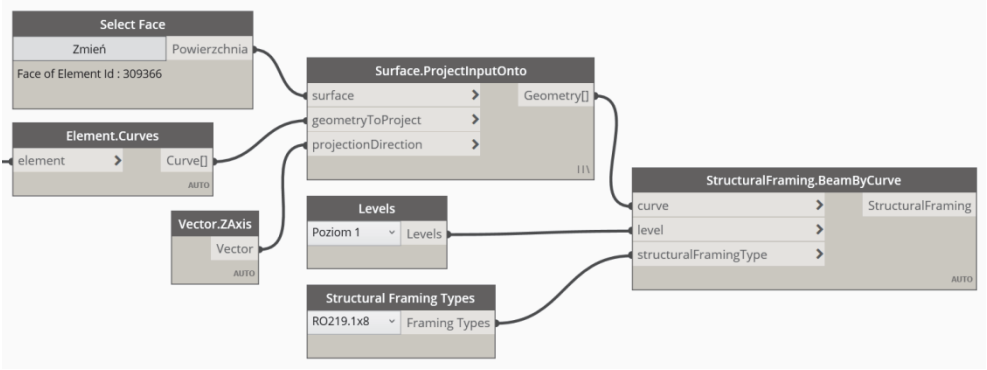
Pierwsza część skryptu wizualnego obejmuje wczytanie elementów prętowych (słupów) i powierzchniowych (dach) modelu Revit za pomocą węzłów *Select Model Elements*. Słupy dodatkowo zostają zamienione na bryły w oparciu o opcję *Element.Solids*. Z kolei linie przewidziane do zamiany na belki nośne dachu zostały zamienione na krzywe typu *curve* za pomocą opcji *Element.Curves* (rys. 5.38).



Rys. 5.38. Import elementów konstrukcyjnych z Revit do Dynamo

Druga część to operacja zamiany i dopasowania linii bazowych jako belek dachowych dzięki węzłowi *StructuralFraming.BeamByCurve*. Aby uzyskać dopaso-

wanie do krzywizny dachu, geometria linii musiała być wcześniej rzutowana na płaszczyznę dachu za pomocą opcji *Surface.ProjectInputOnto* (rys. 5.39).

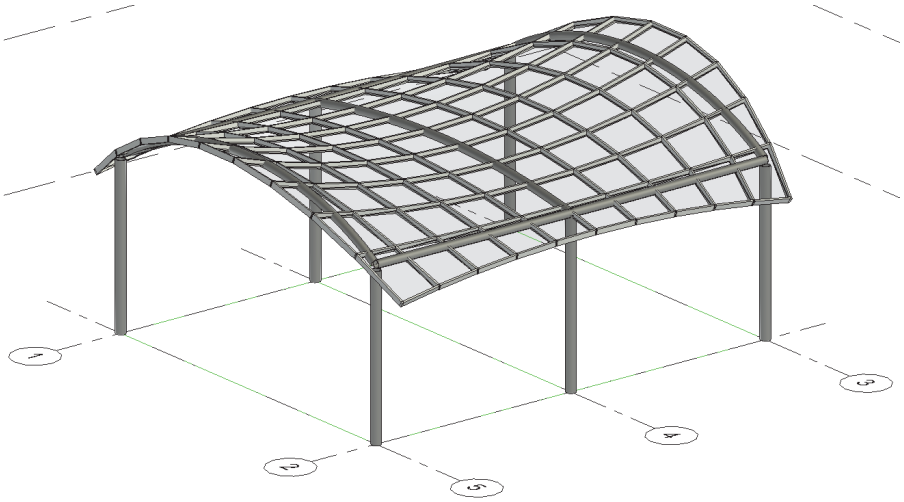


Rys. 5.39. Definicja geometrii dachu

W efekcie zamodelowano w środowisku Dynamo konstrukcję zadaszenia o formie pokazanej na rysunku 5.40.

Rys. 5.40. Konstrukcja zadaszenia przygotowana w Dynamo

Dopełnieniem modelowania konstrukcji zadaszenia jest określenie przekrojów belek dachowych za pomocą węzła *Structural Framing Types* (rys. 5.39), który odczytuje przekroje elementów konstrukcyjnych przyjętych w środowisku Revit. Definiując podkonstrukcję dachu w programie Revit oraz wykonując dopasowanie poszczególnych elementów do siebie (słupy, belki, dach), uzyskuje się zadaszenia o całkiem interesującej formie, pokazanej na rysunku 5.41. Dodatkowo zdefiniowano ruszt i przeszklenie dachu, wykorzystując opcje kształtującej go rodziny Revit.



Rys. 5.41. Model gotowej konstrukcji zadaszenia w środowisku Revit

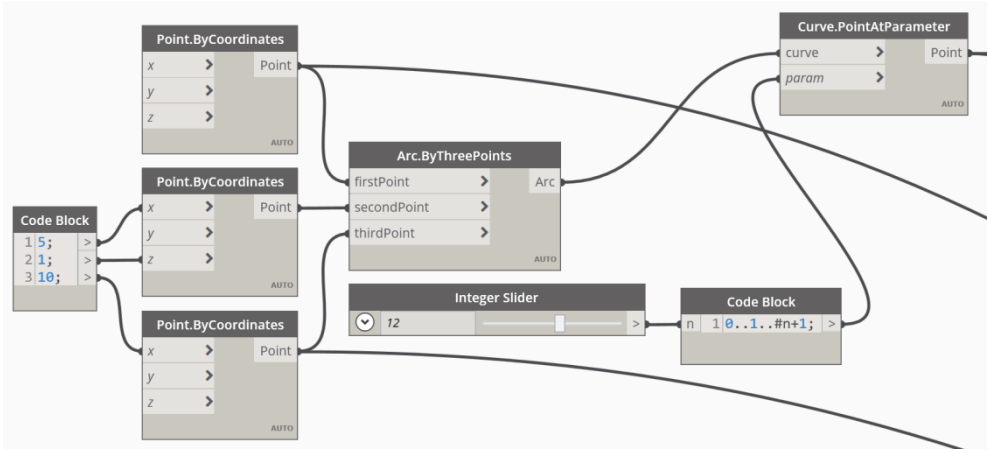
Co istotne, jego geometria, jak i inne informacje kodowane w modelu BIM mogą być na bieżąco aktualizowane dzięki współpracy modeli Revit – Dynamo, jak i rozwijane o kolejne opcje czy komponenty.

5.8.2. Współpraca z Autodesk Robot Structural Analysis

Drugim przykładem integracji środowiska programowania wizualnego Dynamo jest wspomniane już włączenie go jako dodatku do jednego z powszechnie stosowanych programów służących obliczeniom inżynierskim Autodesk Robot Structural Analysis 2022. Fakt tej integracji świadczy zarówno o poszukiwaniu dróg rozwoju projektowania zintegrowanego, jak również coraz większej świadomości projektantów w zakresie projektowania parametrycznego. Analogicznie jak w przypadku środowiska Revit użytkownik ma możliwość równoległej pracy w Dynamo i Robocie oraz interakcji pomiędzy skryptem (modelem generowanym w Dynamo) a modelem numerycznym w Robocie.

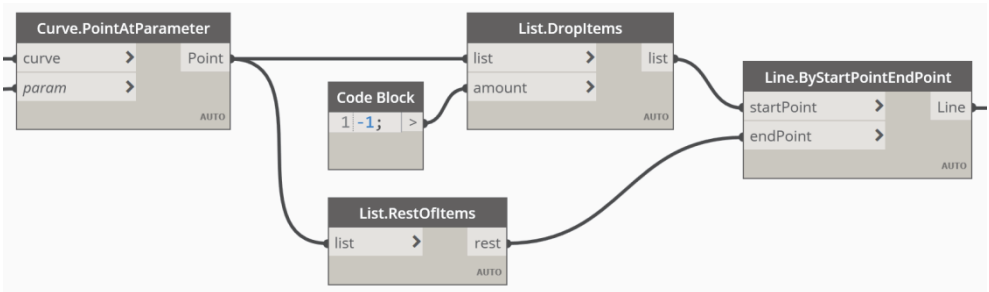
Jako przykład zaprezentowany zostanie skrypt, za pomocą którego wygenerowano model łuku nośnego oraz jego model obliczeniowy (MES), zawierający warunki brzegowe (podpory), przypadki i obciążenia, przekroje, oraz uruchomiono jego analizę statyczną. Całość kodowana jest w skrypcie wizualnym i po uruchomieniu generowana bezpośrednio w Robocie.

Pierwszy etap to modelowanie łuku za pomocą węzła *Arc.ByThreePoints*. Analogicznie jak w poprzednich przykładach wygenerowano punkty podziału za pomocą opcji *Curve.PointAtPArameter*, tak aby pomiędzy nimi zamodelować elementy obliczeniowe MES (rys. 5.42).



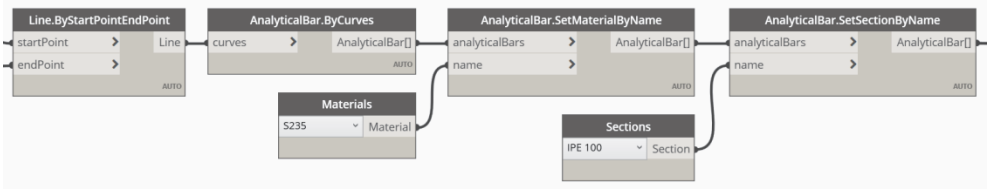
Rys. 5.42. Definicja geometrii łuku w środowisku Dynamo

Elementy te modelowano w oparciu o węzeł *Line.ByStartPointEndPoint*, który, jak już wiadomo, jest jednym z podstawowych w tym zakresie (rys. 5.43).



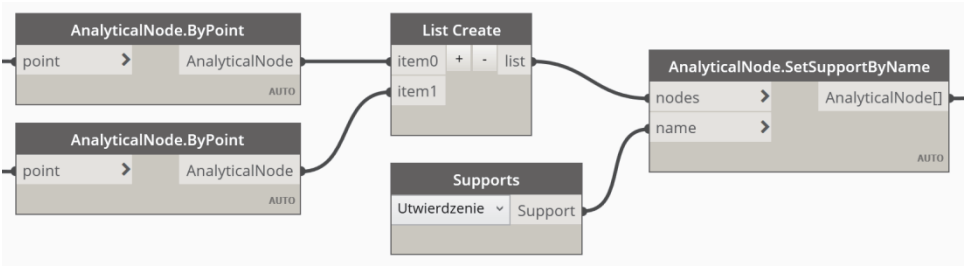
Rys. 5.43. Dyskretyzacja łuku w środowisku Dynamo

Mając wygenerowane elementy – poszczególne sekcje łuku, można zacząć definicję ich właściwości obliczeniowych, tak aby posiadały one cechy elementów MES. W tym celu nadano im przede wszystkim atrybuty elementów obliczeniowych za pomocą węzła *AnalyticalBar.ByCurves* (rys. 5.44). Cechy materiałowe i przekroje elementów określono, wykorzystując odpowiednio opcje *AnalyticalBar.SetMaterialByName* i *AnalyticalBar.SetSectionByName* (rys. 5.44).



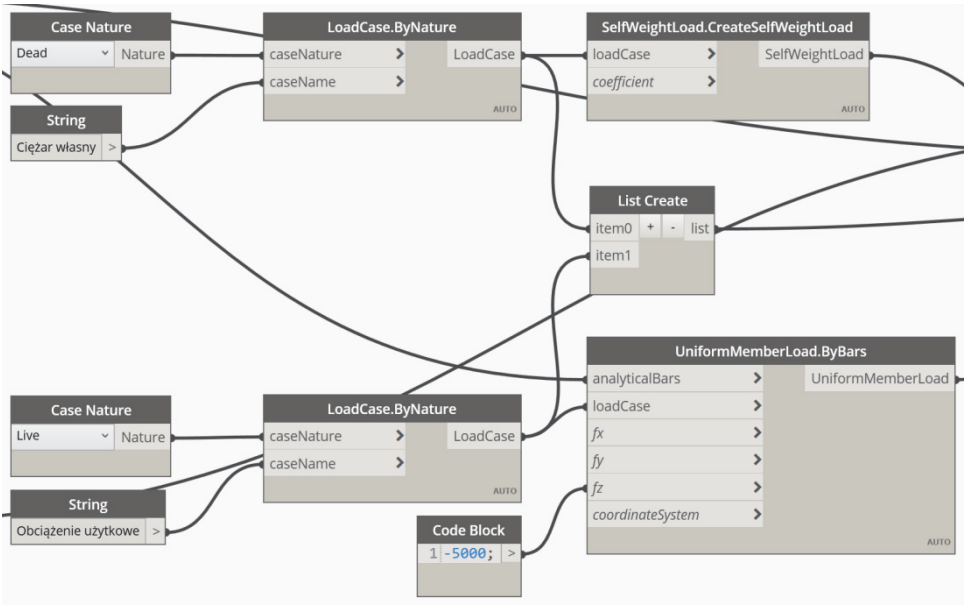
Rys. 5.44. Definicja charakterystyk elementów łuku

Kolejna część skryptu to definicja podpór. Wykorzystano tutaj węzeł *Analytical Node.SetSupportByName*. Aby zdefiniować podpory, konieczne było określenie danych wejściowych w postaci listy węzłów, w których zdefiniowane są podpory, oraz określenie ich typów. Zastosowano opcje typu *AnalyticalNode.ByPoint* i *Supports* (rys. 5.45). Dany typ (rodzaj) podpór określony jest w środowisku Robot i jest z niego wczytywany.



Rys. 5.45. Definicja podpór łuku

Kolejny etap to definicja przypadków obciążeń. Służy do tego węzeł *LoadCase. ByNature*, a użytkownik ma do dyspozycji przypadki predefiniowane w środowisku Robot za pomocą opcji *Case Nature*. W niniejszym przykładzie zdefiniowano przypadek obciążenia ciężarem własnym *Dead* i obciążenie użytkowe *Live* (rys. 5.46).

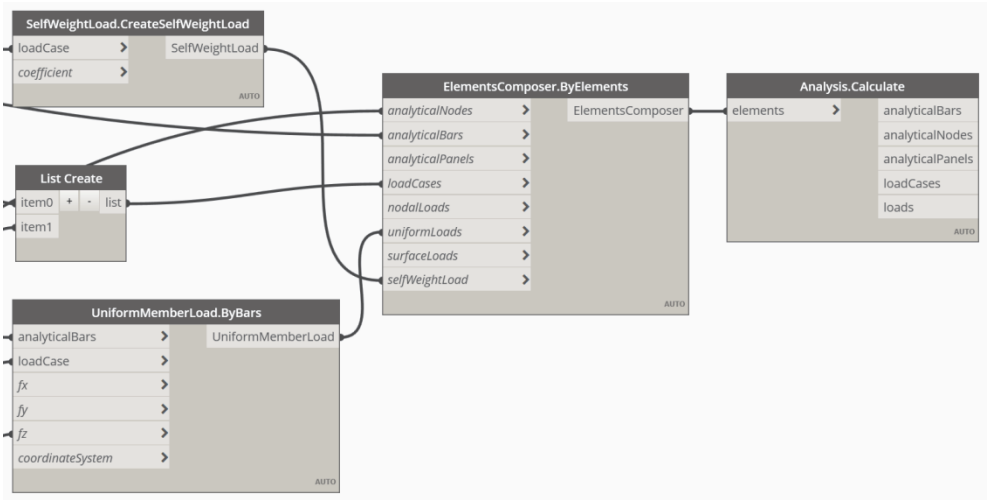


Rys. 5.46. Określenie przypadków i wartości obciążeń

Ich wartości są w tym przypadku definiowane odpowiednio za pomocą opcji *SelfWeightLoad.CreateSelfWeightLoad* i *UniformMemberLoad.ByBars*. Użytkow-

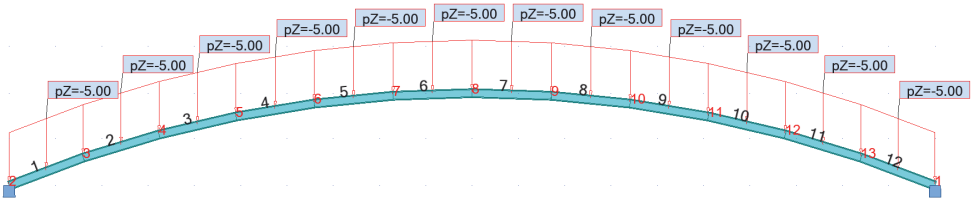
nik ma możliwość definiowania podstawowych rodzajów obciążeń dostępnych w środowisku Robot.

Ostatnia faza to uruchomienie obliczeń. Służy do tego węzeł *ElementsComposer.ByElements* (rys. 5.47). W węźle tym są zbierane wszystkie informacje wsadowe niezbędne do scalenia modelowanej struktury w modelu MES i uruchomienia obliczeń.



Rys. 5.47. Faza uruchomienia obliczeń

Finalnie użytkownik otrzymuje w środowisku Robot model widoczny na rysunku 5.48, gdzie na podstawie aktualnych obliczeń może dokonać analizy statycznej projektowanej konstrukcji, a także przeprowadzić jej wymiarowanie.



Rys. 5.48. Model MES łuku w środowisku Robot

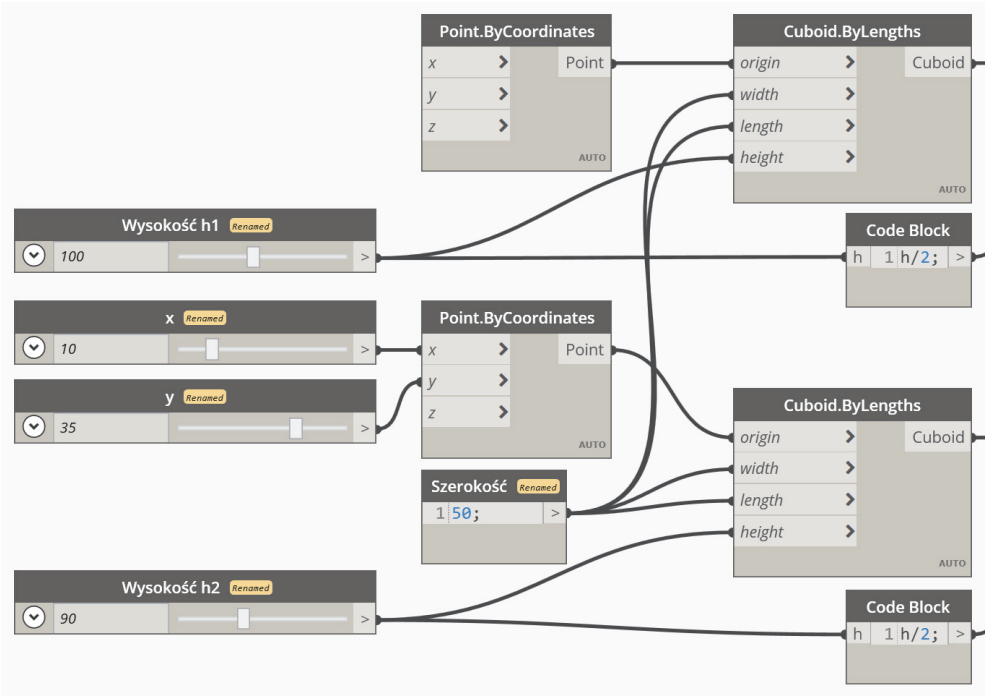
5.9. PROJEKTOWANIE GENERATYWNE

Tak jak to opisano na początku pracy, wraz z rozwojem komputeryzacji procesu projektowania rozwijano coraz to bardziej zaawansowane techniki, co m.in. doprowadziło do parametryzacji modelowania obiektów budowanych. Ciekawym kierunkiem i opcją jest tutaj projektowanie generatywne, które pozwala na uzyskanie w automatyczny sposób wielu wariantów rozwiązań. Dzięki temu inżynier projektant, w efekcie także inwestor mają możliwość optymalizacji całej inwestycji

zgodnie z założonymi kryteriami. W niniejszym podrozdziale przedstawiony zostanie praktyczny przykład zastosowania projektowania generatywnego w oparciu o programowanie wizualne.

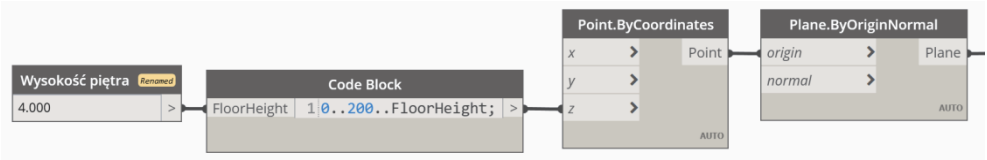
Przykład obejmuje analizę i optymalizację powierzchni projektowanych dwóch budynków. Wykorzystana zostanie w tym celu funkcja projektowania generatywnego dostępna jako dodatek do środowiska Revit i Dynamo. Użyty został tu przykładowy skrypt załączony w pakiecie instalacyjnym, który poddano odpowiedniej modyfikacji.

Skrypt wizualny rozpoczyna się od definicji parametrów określających geometrię, wymiary dwóch budynków o bryłach prostopadłościennych. Zostały one modelowane przy wykorzystaniu węzłów *Cuboid.ByLengths*, które zostały połączone w jedną bryłę za pomocą opcji *Solid.ByUnion* (rys. 5.49).



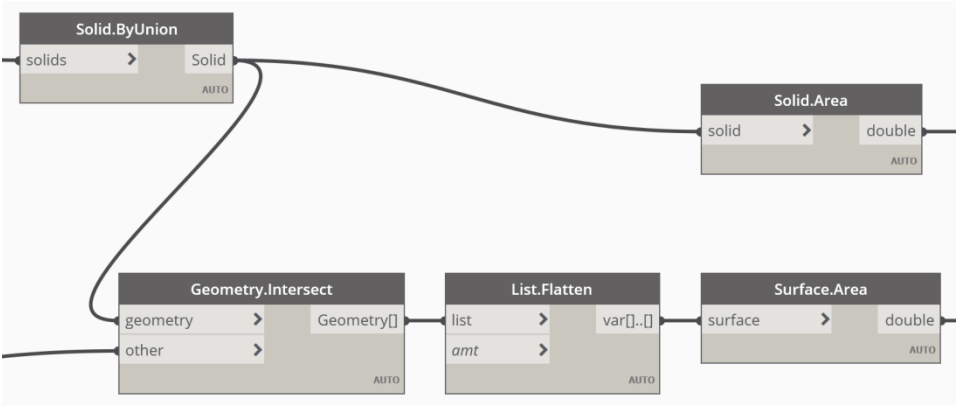
Rys. 5.49. Definicja parametrów sterujących i brył budynków

W budynkach zdefiniowane zostały stropy za pomocą węzła *Plane.ByOrigin Normal* (rys. 5.50).



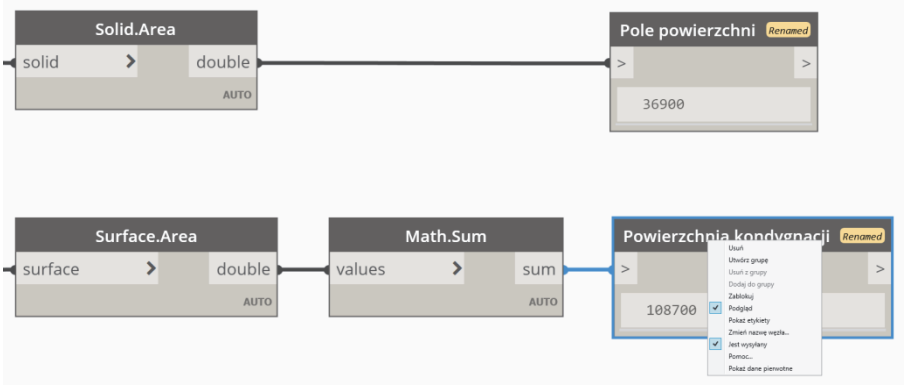
Rys. 5.50. Definicja stropów budynków

Druga część skryptu to określenie parametrów i kryteriów analizy generatywnej. Parametrami, które będą używane jako kryteria optymalizacji, są powierzchnia budynków i powierzchnia wszystkich stropów. W pierwszym przypadku powierzchnia określona została za pomocą opcji *Solid.Area*. Powierzchnia stropów została zsumowana za pomocą węzła *Surface.Area*. Następnie wyznaczono sumę szeregu za pomocą opcji *Math.Sum* (rys. 5.51).



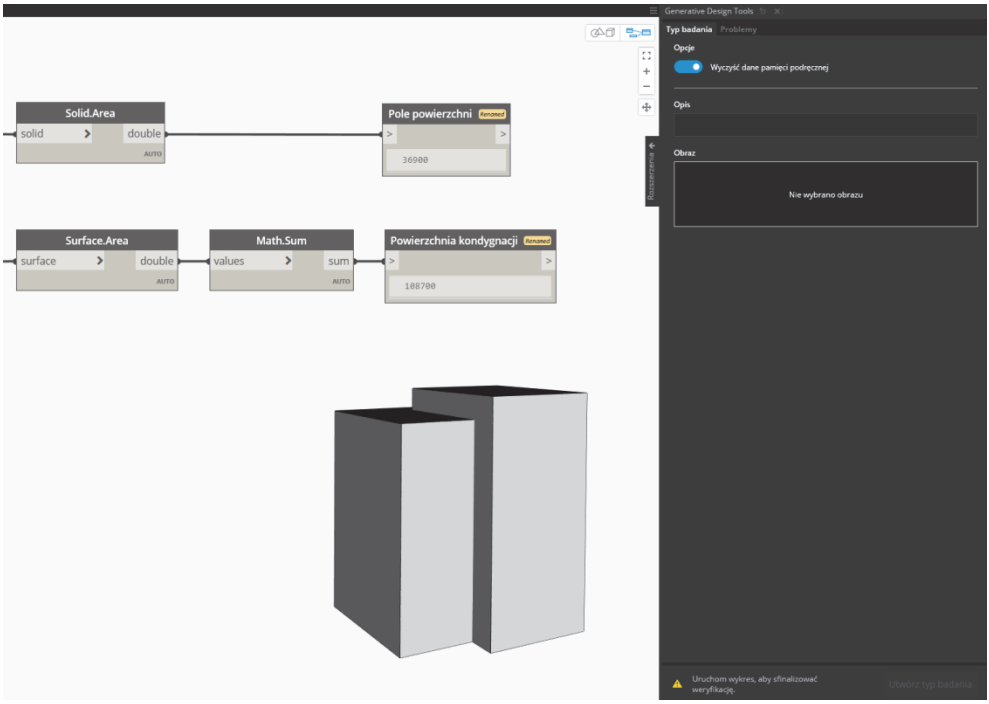
Rys. 5.51. Określenie parametrów optymalizacji generatywnej

Parametry te, tj. powierzchnia budynków i powierzchnia wszystkich stropów, zdefiniowane zostały jako parametry wynikowe analizy, co wymagało oznaczenia ich jako *Jest wysyłany* (rys. 5.52).



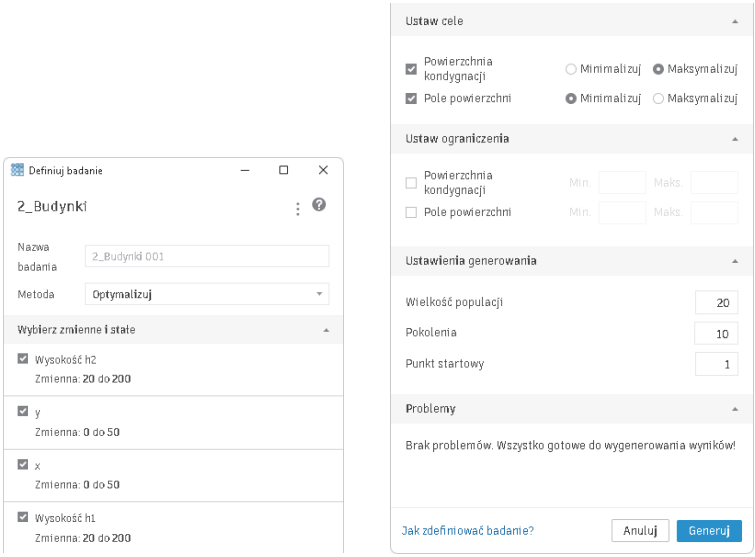
Rys. 5.52. Definicja parametrów wynikowych analizy

Tak przygotowany skrypt stanowi wsad do przeprowadzenia zasadniczej analizy generatywnej. W środowisku Dynamo służy do tego narzędzie *Projektowanie generatywne*, doinstalowane do jego standardowej instalacji. W pierwszym kroku uruchamiana jest opcja *Otwórz narzędzia projektowania generatywnego*. Po uruchomieniu skryptu otrzymuje się komunikat o gotowości do przeprowadzenia zasadniczej części analizy generatywnej (5.53).



Rys. 5.53. Uruchomienie analizy generatywnej w środowisku Dynamo

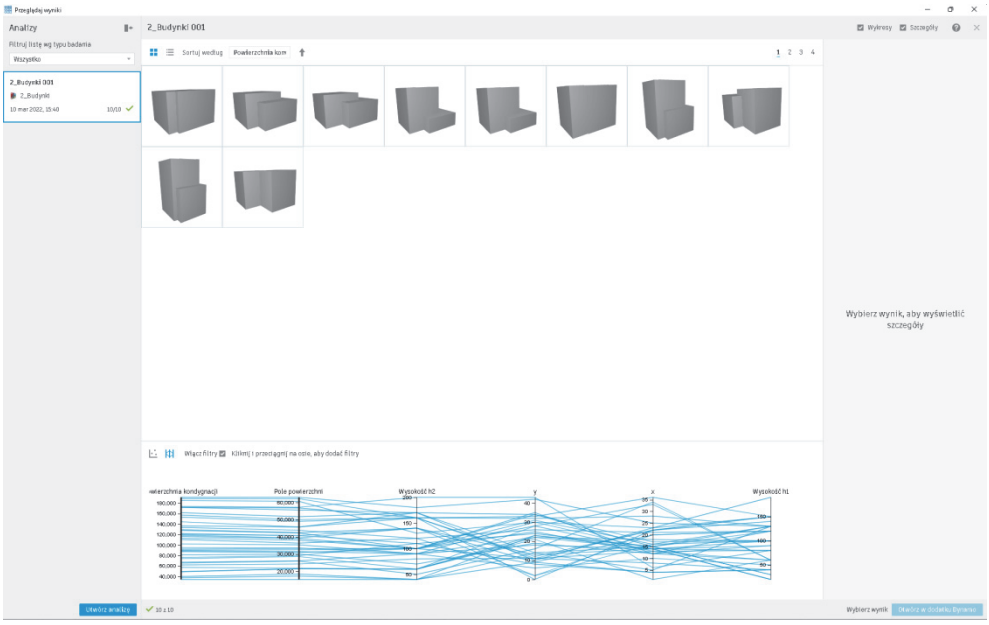
Następnie należy uruchomić i przygotować analizę za pomocą opcji *Utwórz badanie*. Użytkownik zobowiązany jest podać ścieżkę dostępu do pliku wsadowego – skryptu wizualnego i może przejść do określania parametrów analizy generatywnej. Składa się ona z dwóch części: określenia zmiennych i stałych (rys. 5.54a) i określenia celów i ograniczeń (5.54b). W tym przypadku stałymi i zmiennymi są parametry zdefiniowane w skrypcie Dynamo jako wysokości i położenia pól x i y. Cele, czyli kryteria optymalizacji, określono jako minimalizację powierzchni budynków przy jednoczesnej maksymalizacji powierzchni wszystkich kondygnacji. Tak przygotowana analiza została uruchomiona za pomocą opcji *Generuj*. Użytkownik ma do dyspozycji kilka opcji generowania wyników.



a) b)

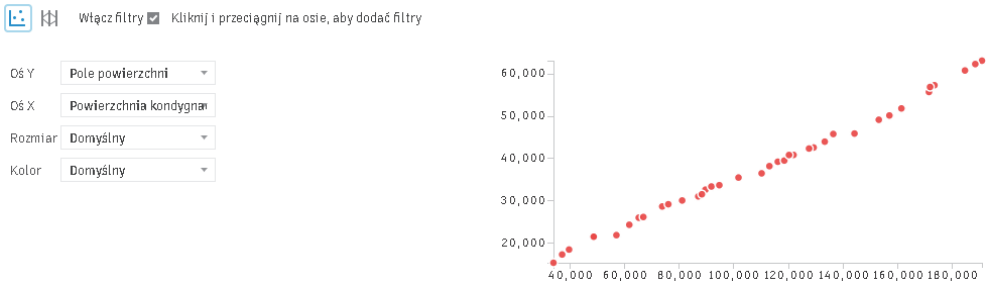
Rys. 5.54. Określenie parametrów analizy generatywnej: a) określenie zmiennych i stałych, b) określenie celów i ograniczeń

W wyniku przeprowadzonej analizy uzyskano 10 różnych wariantów budynków (rys. 5.55). Użytkownik ma możliwość filtrowania (zawężania zakresu parametrów wsadowych), a tym samym uzyskania wyników odfiltrowanych.



Rys. 5.55. Wyniki analizy generatywnej

Możliwa jest także analiza zależności pomiędzy analizowanymi parametrami, co pokazano przykładowo na rysunku 5.56.



Rys. 5.56. Analiza zależności pomiędzy uzyskanymi parametrami analizy generatywnej

6 MOŻLIWOŚCI PRAKTYCZNYCH ZASTOSOWAŃ SYSTEMÓW VPL W INŻYNIERII

W poprzednich rozdziałach szerzej zaprezentowano środowisko programistyczne Dynamo jako jedno z podstawowych w kategorii inżynierskiego modelowania obiektów budowlanych. Obecnie jest to jedna z najlepszych i najpowszechniej stosowanych platform programowania wizualnego do kształtowania konstrukcji inżynierskich. Tak jak to przedstawiono we wstępie, środowisk takich jest więcej, a kilka z nich znajduje profesjonalne zastosowanie, i jest w dalszym ciągu rozwijanych. W niniejszym rozdziale przedstawione zostaną obszary i przykłady praktycznych zastosowań programowania wizualnego w projektowaniu inżynierskim.

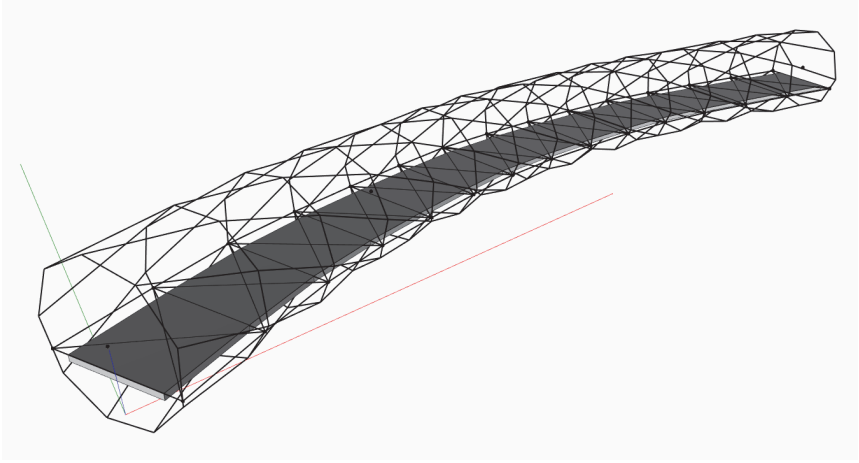
6.1. KSZTAŁTOWANIE FORMY OBIEKTÓW

Wydaje się, że obecnie zastosowanie programowania wizualnego w modelowaniu geometrii projektowanych obiektów to największa zaleta tej technologii. Jest to narzędzie bardzo pomocne w kształtowaniu wszystkich tych geometrii, które z jednej strony dają się łatwo opisać – zdekomponować do prostych figur i elementów geometrycznych, a z drugiej strony wręcz odwrotnie, formy te są na tyle dowolne, że opis ten jest bardzo skomplikowany. Tak jak przedstawiono to w poprzednim rozdziale i pokazano na przykładach, bardzo szerokie zastosowanie mają tutaj elementy geometryczne typu splain i NURBS, gdyż pozwalają one na dowolną edycję geometrii końcowej za pomocą punktów sterujących. Łatwość tej edycji i jej dynamika pozwala na przygotowanie wielu wariantów projektowanych obiektów.

Osobną kategorią jest kształtowanie form organicznych. Nie dotyczy to jedynie budynków, bowiem wiele obiektów inżynierskich takich jak mosty, kładki dla pieszych ma formę bioniczną, często odwołującą się do świata roślin czy zwierząt, a nawet do najgłębszych pokładów struktury tkanki, jakim jest choćby DNA. Dobrym przykładem może być kładka dla pieszych Helix Bridge w Singapurze (rys. 6.1), której struktura została opisana dwiema krzywymi helikoidalnymi. Helikoida to powierzchnia powstająca na skutek jednostajnego obrotu prostej dookoła nieruchomej osi, która jednocześnie przesuwa się równolegle do tej osi. Powierzchnia tego typu dość dobrze modeluje kształt spiral ludzkiego DNA.

Rys. 6.1. Kładka dla pieszych Helix Bridge w Singapurze [98]

Z jednej strony opis tego typu struktur może być ciągły za pomocą funkcji matematycznych, a z drugiej płynny, elastyczny, jak w przypadku użycia krzywych typu splain czy też powierzchni NURBS. Bardzo dobrym rozwiązaniem jest zatem zastosowanie właśnie modelowania opartego na programowaniu wizualnym, które daje wszystkie te możliwości. Na rysunku 6.2 pokazano model kładki Helix Bridge powstały w środowisku Dynamo.



Rys. 6.2. Model kładki dla pieszych o strukturze helikoidalnej wykonany w środowisku Dynamo

Kształtowanie formy obiektów przy zastosowaniu programowania wizualnego jest także efektywne w przypadku projektowania powtarzających się wypełnień, które funkcjonują modularnie w projektowanej strukturze obiektu.

6.2. KSZTAŁTOWANIE KONSTRUKCJI

Projektowany obiekt o ustabilizowanej formie w naturalny sposób musi być oparty na strukturze nośnej, która niejako stanowi jego szkielet. W tym wypadku programowanie wizualne ma zastosowanie w modelowaniu czystej konstrukcji. Projektant ma do dyspozycji szereg opcji, które pozwalają na definiowanie zarówno podstawowych elementów, jak i całych układów konstrukcyjnych. Efektywne w tym zakresie są pakiety automatyzujące modelowanie najczęściej wykorzystywanych struktur nośnych.

Mając dobrany układ konstrukcyjny projektowanego obiektu, podstawową zaletą programowania wizualnego jest możliwość w zasadzie dowolnego kształtowania i dopasowania geometrii konstrukcji. Sposób modelowania jest podobny jak w przypadku kształtowania ogólnej formy obiektu. Często jednak konstrukcja nośna obiektu wykorzystuje np. elementy osłonowe do generacji poszczególnych elementów konstrukcyjnych. W takiej sytuacji elementy konstrukcyjne modelowane są w nie w pierwszym, ale w dalszym etapie. Wykorzystywane są tu właśnie obiekty stanowiące „obudowę" projektowanego obiektu takie jak ściany, dach itp. Elementy konstrukcyjne są definiowane w wyniku operacji przeprowadzanych na elementach już definiowanych. Jest to bardzo duże ułatwienie, bo pozwala na rozmieszczenie struktury konstrukcyjnej i dopasowanie jej do geometrii często niesymetrycznych, stanowiących powierzchnie dowolne.

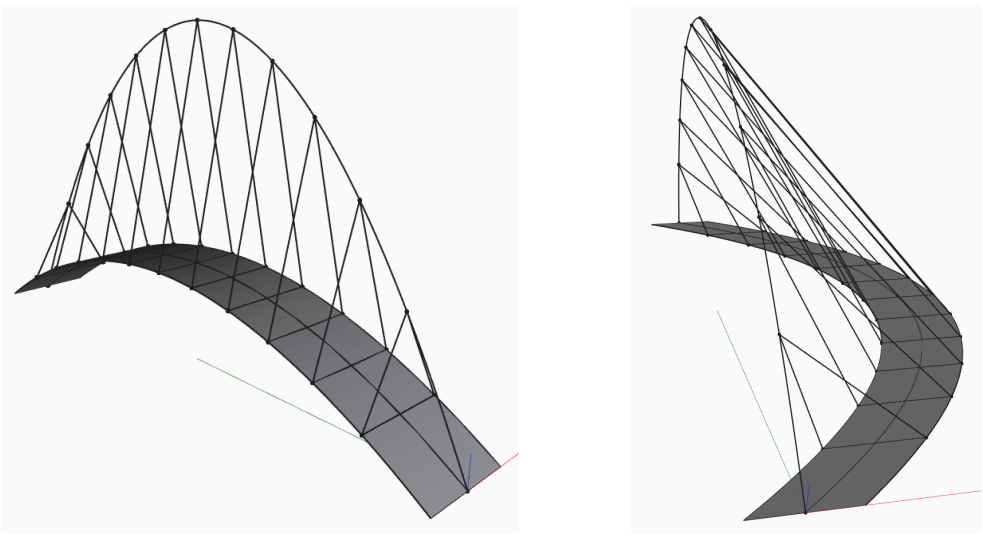
Kolejną możliwością, która jest tutaj stosowana, jest generacja rozmaitych układów nośnych w sposób dynamiczny, np. stosując możliwość elastycznie zmieniających się wymiarów konstrukcji czy podziałów modularnych. W ogóle oparcie geometrii projektowanego obiektu na siatce punktów węzłowych i stosowanie podejścia dynamicznego stanowi zasadniczą różnicę w stosunku do klasycznej koncepcji i możliwości CAD.

6.3. PROJEKTOWANIE OBIEKTÓW MOSTOWYCH

Z uwagi na specyfikę całej branży projektowanie obiektów inżynierskich to w zasadzie osobna kategoria. Przede wszystkim należy zwrócić tu uwagę na podstawowy podział obiektów tego typu na mosty i wiadukty, których układy konstrukcyjne są w zasadzie bardzo podobne, natomiast naturalne różnice są w sposobie posadowienia i warunkach gruntowych, oraz kładki dla pieszych, których konstrukcja i stosowane układy statyczne to podbranża mostownictwa.

Na ich przykładzie przedstawiono możliwości przeprowadzenia analizy wariantowej w zakresie kształtowania formy i geometrii w oparciu o programowanie wi-

zualne. Widok dwóch propozycji kładki dla pieszych o konstrukcji podwieszonej pokazano na rysunkach 6.3a i 6.3b.



a) b)

Rys. 6.3. Warianty kładki dla pieszych

W pierwszym przypadku zdecydowano się na wyniesienie pomostu i niższy symetryczny łuk. W wariancie drugim dokonano obniżenia pomostu, wygięcia go w planie oraz przesunięcia wierzchołka łuku nośnego w stosunku do osi kładki. Wszystkie te operacje dokonane były dynamicznie, w oparciu o parametryczną definicję punktów sterujących geometrią pomostu i łuku. Parametry te były definiowane za pomocą węzłów typu *slider*, które umożliwiają niejako deformowanie geometrii obiektu w czasie rzeczywistym.

Należy zwrócić również uwagę na możliwości modelowania i kształtowania najbardziej efektywnych konstrukcji mostowych, pozwalających na pokonywanie większych rozpiętości, tj. obiektów wantowych i wiszących. Jest to szczególnie istotne w kontekście możliwości, jakie daje projektowanie choćby mostów wielkich rozpiętości o konstrukcji wiszącej, przy wykorzystaniu programowania wizualnego, mając na uwadze wyzwania, jakie są w związku z pomysłami realizacji obiektów o rozpiętościach jednego przęsła ponad 3 km!

6.4. ZASTOSOWANIA BIM, INTEROPERACYJNOŚĆ

Tak jak to opisano we wprowadzeniu, programowanie wizualne to jedno z podstawowych narzędzi używanych w technologii BIM. Sukcesywnie wprowadzane do kolejnych środowisk modelowania i programów z nimi związanych (detalowa-

nie, obliczenia itd.) zaczyna stanowić jeden z ich podstawowych dodatków i składników. Ma to miejsce w związku z dwoma głównymi obszarami integracji:

- kształtowanie geometrii obiektów,
- zarządzanie informacją.

Oba te obszary są ze sobą połączone w większym lub mniejszym stopniu, w zależności od specyfiki projektowanego obiektu. W przypadku np. budynku integracja ta może być większa z uwagi na fakt możliwości kształtowania geometrii i zarządzania kluczowymi parametrami obiektu. W przypadku wyizolowanej konstrukcji, np. konstrukcji wsporczej, większym wspomaganiem będzie zastosowanie programowania wizualnego do kształtowania geometrii.

6.5. PROJEKTOWANIE PARAMETRYCZNE

Choć sama idea programowania wizualnego oparta jest na parametryzacji, to może ona być realizowana w większym lub mniejszym stopniu. Daną konstrukcję można przecież opisać współrzędnymi punktów, które będą wprowadzone na sztywno, lub zastosować opis matematyczny, gdzie współrzędne tych punktów będą generowane zgodnie z danym wzorem. To drugie podejście daje bardzo duże możliwości kształtowania formy projektowanego obiektu, a także kształtowania jego wszystkich komponentów, w tym oczywiście konstrukcji. Sparametryzowanie, tak jak to pokazano na przykładach, pozwala inżynierowi na opis dyskretny, gdzie struktura jest dzielona na geometrie prymitywne, ale można również zastosować opis ciągły, za pomocą bardzo dużej bazy funkcji matematycznych. To niejako próba opisu świata za pomocą matematyki, gdzie dla każdej geometrii można określić funkcję ją opisującą. Ten walor ma znaczenie z jednej strony utylitarne, czysto techniczne, a z drugiej konceptualne, gdzie dana struktura odzwierciedla pewien świat niefizyczny, będący opisem ściśle abstrakcyjnym. Warto o tym pamiętać, stosując projektowanie sparametryzowane, które to rozwija wielu twórców, designerów, architektów.

6.6. PROJEKTOWANIE GENERATYWNE

Możliwości, jakie daje komputeryzacja procesu projektowania, w naturalny sposób ujawniają coraz szersze zakresy jej zasięgu, pozwalające na przygotowywanie wielu wariantów oraz analizę ich zalet i wad. Projektowanie generatywne to naturalny etap rozwoju w tym zakresie, który z sukcesem uzupełnia i rozszerza opcje projektowania inżynierskiego [99]. Obejmuje ono coraz to szersze obszary zastosowań, w tym także w architekturze, ale co istotne również w projektowaniu konstrukcyjnym [100].

Postawione zagadnienie optymalizacyjne może być opisane przez skrypt wizualny, który następnie jest uruchamiany i wykorzystywany przez specjalnie do tego przeznaczone programy. Przykładowo Revit oferuje opcję pod nazwą *Projektowanie generatywne*, które zostało omówione w podrozdziale 5.9, gdzie zaprezentowa-

no przypadek optymalizacji wielkości budynków. Użytkownik ma zatem możliwość zoptymalizowania procesu projektowania w opisanych zakresach, jak również przygotowania autorskich skryptów i dostosowania ich dla swoich indywidualnych potrzeb. Wydaje się, że programowanie generatywne, oprócz prac ściśle konceptualnych i studiów, w niedługim czasie może być jednym z podstawowych narzędzi stosowanych w zwykłej praktyce inżynierskiej. Zależy to w dużej mierze od dołączania tych narzędzi do najpopularniejszych środowisk projektowych oraz świadomości i poziomu wykształcenia inżynierów projektantów, o czym traktuje ostatni rozdział niniejszej publikacji.

6.7. MOŻLIWOŚCI INNYCH ZASTOSOWAŃ SYSTEMÓW VPL

Opisane w niniejszym rozdziale obszary zastosowań programów wspomagających projektowanie wykorzystujących programowanie wizualne to te, które są aktualnie najlepiej zagospodarowane. Nie są oczywiście one jedyne. Środowiska programistyczne typu VPL znajdują zastosowanie tam, gdzie można zaobserwować i uchwycić zależności, jakie zachodzą pomiędzy elementami składowymi obiektu będącego przedmiotem zagadnienia. Tak naprawdę programowanie wizualne można zastosować wszędzie tam, gdzie możliwy jest opis danego problemu w sposób mniej lub bardziej sparametryzowany. Można zaryzykować stwierdzenie, że cały świat można opisać matematycznie, a więc i cały świat można sparametryzować. Tak naprawdę tylko i wyłącznie od wyobraźni inżyniera projektanta programisty zależy, jak i gdzie zastosuje on środowisko Grasshopper, Dynamo czy inne im podobne. Tak jak można tworzyć muzykę na przykład w oparciu o ciąg Fibonacciego, tak i zakodować można dowolny obiekt fizyczny, proces lub zjawisko. Ma to miejsce, o czym można się przekonać, przeglądając *case studies* z zakresu programowania wizualnego i to niekoniecznie z branży architektonicznej czy budowlanej.

7 KIERUNKI ROZWOJU PROJEKTOWANIA OPARTEGO NA PROGRAMOWANIU WIZUALNYM

Omawiając dalsze kierunki rozwoju programowania wizualnego pod kątem aplikacji w projektowaniu inżynierskim, należy podkreślić, że w tym momencie odpowiadają one ścieżkom ich głównych zastosowań, opisanych w poprzednim rozdziale. Obecnie stosowane narzędzia są wciąż rozwijane i ulepszane oraz implementowane do różnych środowisk.

Analizując natomiast obszary przyszłych zastosowań, należy zacząć od wniosku ogólnego. Dotyczy on generalnego kierunku, w jakim projektowanie oraz całościowy zakres realizacji inwestycji idą. Kierunek ten został już dawno wytyczony i jest nim coraz powszechniejsza i intensywniejsza komputeryzacja, cyfryzacja, digitalizacja i automatyzacja całego procesu. Projektowanie oparte na programowaniu wizualnym jest jednym z komponentów, który może nie jest w tym momencie niezbędny, bo zawsze przecież można je zastąpić tradycyjnym podejściem stosowanym z sukcesem do tej pory, ale wydaje się, że programowanie wizualne będzie w przyszłości nabierało coraz to większego znaczenia. W niedługiej już może perspektywie pewnego rodzaju obiekty będą w dużej mierze projektowane i realizowane z dużym udziałem właśnie modelowania opartego na programowaniu wizualnym, które, jak można przewidywać, stanie się w pewnym przypadkach właśnie podstawową opcją. To pierwszy, ogólny wniosek. Pozostałe zostały przedyskutowane poniżej, określając również nie tylko kierunki rozwoju omawianej technologii, ale również i jej potrzeby.

7.1. EDUKACJA NA POZIOMIE ELEMENTARNYM

Obserwując od wielu już dziesięcioleci rozwój techniki projektowania oparty na komputeryzacji i cyfryzacji, nie sposób zacząć dyskusji od tematu edukacji w tym zakresie. Choć może się wydawać, że nie jest to element ściśle związany z omawianym tematem, to przecież właśnie od świadomości i poziomu wykształcenia inżynierów wielu branż zależy powszechność stosowania systemów wspomagania projektowania. Najistotniejsza tutaj jest edukacja obejmująca systemy już wdrożone, które niejako stają się niepisanym standardem, jak przykładowo system CAD, a także środowiska i systemy aktualnie rozwijane i wdrażane stopniowo, jak choćby technologia BIM. Od poziomu edukacji w zakresie wdrażania nowoczesnych technologii wspomagania projektowania zależy ich rozwój i upowszechnianie w praktyce inżynierskiej. W odniesieniu do programowania wizualnego należy stwierdzić, że w krajowym systemie dydaktycznym przyjętym na wydziałach kształcących inżynierów budowlanych nie jest to przedmiot powszechny i łatwy do zrealizowania. Podstawowym problemem jest tutaj fakt, że nie funkcjonuje on na wszystkich ścieżkach dydaktycznych, a jedynie na wybranych, najczęściej związa-

nych z technologią BIM lub pokrewnych. Kolejnym problemem jest pewnego rodzaju bariera merytoryczna, a może bardziej mentalna, związana z podejściem studentów budownictwa do programowania jako takiego. Co prawda w pierwszych semestrach powszechną praktyką jest nauka jednego z podstawowych języków programowania, jednakże student, przyszły inżynier budowlany programista to niezmierna rzadkość. W tym miejscu należy zauważyć jeszcze jedno zjawisko i pewnego rodzaju sprzeczność. Nauka programowania jest realizowana przecież już od szkoły podstawowej, i to na poziomie wczesnoszkolnym. W ostatnich latach edukacji podstawowej uczniowie zapoznają się ze środowiskiem i językiem programowania C+. Także w szkołach średnich w ramach przedmiotu informatyka programowanie jest także nauczane. Skąd zatem tak duże problemy i opory studentów budownictwa w tym zakresie? Odpowiedź na to pytanie jest złożona i wykracza daleko poza zakres niniejszego rozdziału, natomiast dyskutowane problemy można bardzo łatwo powiązać z obniżeniem się poziomu nauczania w zakresie nauk ścisłych, brakiem nauczania logiki i ogólnym brakiem chęci zdobywania wiedzy i umiejętności jako takich. Podsumowując więc, te dwa elementy to podstawowe bariery, które skutecznie ograniczają przyszłe możliwości wykorzystania programowania wizualnego w praktyce inżynierskiej.

Bariery te oczywiście są do pokonania. Co jest konieczne dla kierunków rozwoju programowania wizualnego – upowszechnienia go w środowisku przyszłych, jak i funkcjonujących inżynierów. Proces ten może nieco powoli, ale jednak zachodzi, bo na kilku przynajmniej uczelniach przedmiot obejmujący tematykę zastosowania programowania wizualnego w projektowaniu oraz praktyczne ćwiczenia mające na celu naukę tej technologii są realizowane.

Osobną ścieżką edukacji są szkolenia. W ostatnim okresie można zauważyć zintensyfikowanie działań na tym polu. Przede wszystkim sukcesywnie poszerzana jest oferta w zakresie szkoleń z wielu obszarów obejmujących umownie nazywane komputerowe wspomaganie projektowania. Jednocześnie zauważa się wzrost liczby firm czy instytucji prowadzących szkolenia. Obejmują one także tematykę programowania wizualnego, głównie w środowisku Dynamo wraz z integracją ze środowiskiem Revit. Dotyczy to zarówno samej idei programowania wizualnego zorientowanego na modelowanie projektowanych obiektów budowalnych, jak i „oprogramowanie" podbranż, np. infrastruktury. Patrząc na rozwój tego obszaru, należy przede wszystkim pozytywnie podkreślić kwestię ustawicznego kształcenia, jakiemu podlegają pracujący już w zawodzie inżynierowie, i to nie tylko projektanci. Taka ścieżka rozwoju kompetencji wpisuje się w trendy zachodniego modelu, gdzie po uzyskaniu podstawowego wykształcenia branżowego inżynier (i nie tylko) swoje kompetencje uzyskuje na drodze samodoskonalenia zawodowego w formie szkoleń, studiów podyplomowych, studiów uzupełniających czy wreszcie studiów w innych ścieżkach czy nawet podjęcia trudu pójścia ścieżką doktoryzowania. I to kolejny obszar, a zarazem kierunek, w jakim podążają rozwój i upowszechnienie programowania wizualnego jako takiego w ogóle. Jak już zaznaczo-

no wcześniej, podstawą jakiegokolwiek rozwoju jest m.in. świadomość i upowszechnianie nowoczesnych i innowacyjnych narzędzi technologii, w tym wypadku wydajnie wspomagających i automatyzujących proces projektowania, realizacji, obsługi inwestycji oraz zarządzania obiektem w ciągu jego cyklu życia. Jak również kilkukrotnie wspomniano, programowanie parametryczne, o czym będzie również poniżej w kontekście kierunku rozwoju, daje zupełnie nowe, często abstrakcyjne możliwości wykraczające daleko poza projektowanie inżynierskie. Nie można więc zapominać o tym ważnym aspekcie, jakim jest proces edukacji.

7.2. ROZWÓJ ŚRODOWISK PROGRAMISTYCZNYCH

Korzystnymi czynnikami sprzyjającymi upowszechnieniu zastosowań narzędzi opartych na programowaniu wizualnym, które mogą być używane nie tylko w inżynierii budowlanej, są prace nad ich rozwojem. Środowiska i interfejsy niezbędne do programowania są sukcesywnie rozwijane i aktualizowane. Wystarczy wspomnieć choćby o liczbie wersji rozwojowych programu Dynamo SandBox, który miesięcznie jest aktulizowany na poziomie kilku wersji stabilnych i nawet kilkudziesięciu wersji rozwojowych [101].

Rozwój ten jest prowadzony wielotorowo. Opracowywane i aktualizowane są nowe procedury (węzły) w zakresie rozszerzenia funkcjonalności obejmujących wiele różnorakich obszarów działania. Odnosi się to zarówno do modyfikacji czy tworzenia nowych opcji, jak również tworzenia węzłów. Ten drugi obszar dotyczy przede wszystkim przygotowania pakietów, w których zawarte są węzły przewidziane do wąskiego, specjalistycznego zastosowania w poszczególnych branżach czy nawet ich częściach. Przykładem niech będzie choćby omawiany tutaj pakiet *Structural Analysis for Dynamo*, za pomocą którego w samodzielnej instalacji Dynamo Sanbox można przygotować skrypt, który pozwala na wykonanie sparametryzowanego modelu obliczeniowego konstrukcji, jego uruchomienie oraz uzyskanie wyników w programie Robot Structural Analysis Professional. Odnosząc się do tego konkretnie pakietu, widać konieczność jego rozwoju z uwagi na sam fakt integracji tych dwóch środowisk, jak również braków funkcjonalnych w samym pakiecie. To naturalny kierunek rozwoju programowania wizualnego. Istnieje cały szereg innych pakietów o konkretnym przeznaczeniu, które są uaktualniane w systemie ciągłym (liczba aktualizacji rocznych wynosi nierzadko kilkanaście), oraz projektowane są nowe pakiety. W tym zakresie wydaje się, że w perspektywie kilku następnych lat będzie to podstawowy kierunek rozwoju środowisk programowania wizualnego – rozszerzenie funkcjonalności i oprogramowanie wyspecyfikowanych zagadnień.

Osobnym kierunkiem rozwoju środowisk programistycznych jest ich integracja i implementacja w innych środowiskach czy platformach projektowych. Proces ten zachodzi sukcesywnie i w przeciągu kilku ostatnich lat obserwuje się wprowadzanie i sprzęganie programów programistycznych ze środowiskiem BIM czy MES. System

Dynamo, jak już wspomniano, stał się składnikiem środowisk takich jak Robot, Revit, jak również został włączony do narzędzi służących projektowaniu generatywnemu.

Jednym z efektów wyżej wymienionych prac rozwojowych jest uproszczenie samego procesu programowania dzięki stworzeniu i ulepszaniu środowisk programistycznych. Użytkownik dostaje coraz bardziej wydajne narzędzia, za pomocą których może znacznie skrócić i uprościć proces projektowania.

Niebagatelny jest również darmowy dostęp do niektórych środowisk, jak choćby Dynamo Sandbox, które można pobrać bezpłatnie ze strony [102]. Dodatkowo na stronie https://github.com/DynamoDS/Dynamo [103] udostępniony jest kod źródłowy, który może być wykorzystany przez programistów do opracowywania kolejnych wydań programu. Dzieje się to zresztą bardzo intensywnie, o czym można się przekonać, analizując liczbę wydań wersji rozwojowych dostępnych na stronie https://dynamobuilds.com/ [101]. Jednocześnie środowisko Dynamo jest zintegrowane z wieloma innymi wspomnianymi wcześniej programami znajdującymi zastosowanie w projektowaniu, które oczywiście przeważnie są oferowane odpłatnie. Wydaje się, że takie podejście jest optymalne z punktu widzenia użytkownika i inżyniera, bo z jednej strony ma on darmowy dostęp do wersji środowiska w pełni funkcjonalnego, a z drugiej strony jest ono zintegrowane z najważniejszymi programami, które i tak są oferowane komercyjnie. W przypadku innych aplikacji należy niestety ponieść koszty zakupu licencji oraz późniejszych aktualizacji, co już nie jest tak komfortowe, jak w przypadku programu Dynamo.

Kolejny element niezmiernie istotny w odniesieniu do upowszechnienia opisywanej w pracy technologii programowania wizualnego pod kątem zastosowań inżynierskich to dostęp do dokumentacji. Obejmuje on kwestie obsługi programów, tj. instalacji, działania, rozszerzania funkcjonalności (np. instalacji dodatków), jak również wiedzy w zakresie samego programowania wizualnego. Należy przyznać, że dostępne dokumentacje w tych zakresach są dość obszerne i osoba posiadająca umiejętność samodzielnego wyszukiwania informacji, stosowania i weryfikowania ich w praktyce powinna sobie poradzić z prawidłową obsługą środowisk, jak również być w stanie przygotować podstawowe skrypty. Bazowa wiedza jest z reguły darmowo udostępniana przez producentów oprogramowania, np. [104-106]. Ponieważ omawiane środowiska mają przede wszystkim utylitarne zastosowanie niezbędnym składnikiem dokumentacji są przykładowe pliki – skrypty, określane najczęściej jako *sample*. Baza dostępnych przykładów jest oczywiście sukcesywnie rozszerzana przez skrypty przygotowane przez użytkowników, bądź to prywatnie, bądź przez firmy komercyjne. W tym drugim przypadku należy niestety ponieść pewne koszty, gdyż czasami są one udostępniane odpłatnie. Dopełnieniem darmowej bazy wiedzy są również filmy instruktażowe przygotowywane i udostępniane w najpopularniejszych serwisach wideo dostępnych w internecie. Równolegle funkcjonuje rynek szkoleń oferowanych przez firmy komercyjnie, które czasami organizują darmowe, z reguły krótkie szkolenia, np. w formie webinariów.

Podstawowym natomiast brakiem jest praktycznie zerowa liczba zwartych publikacji naukowych czy naukowo-dydaktycznych w formie podręczników akademickich, gdzie w kompleksowy sposób zaprezentowano by ideę samego programowania wizualnego, jego zastosowania praktycznego, jak również przedstawiono by kilka choćby elementarnych przykładów skryptów, na podstawie których inżynier laik mógłby zacząć pracę w danym środowisku. Ten fakt m.in. stał się głównym impulsem do powstania niniejszej monografii, która, jak autor ma nadzieję, będzie w przyszłości rozwijana i rozszerzana. W literaturze przedmiotowej, głównie zagranicznej, dostępne są choćby opracowania typu [107, 108], jednakże ich forma i zakres nie jest adekwatny z punktu widzenia przedstawionego powyżej. Dlatego można stwierdzić, że jest wyraźny brak tego typu publikacji. W odniesieniu do zastosowania programowania wizualnego w architekturze, jak również samej idei architektonicznego projektowania parametrycznego i algorytmicznego należy przywołać bardzo dobre publikacje cytowane na początku niniejszej pracy, a mianowicie [44] i [56]. Tak jak już stwierdzono, brak jest natomiast publikacji tego typu odnoszących się do kształtowania i modelowania struktur konstrukcji w oparciu o programowanie wizualne. Jest to zatem jedna z potrzeb i kierunków przyszłych prac.

7.3. INTEGRACJA ŚRODOWISK PROGRAMISTYCZNYCH I PROJEKTOWYCH

Naturalną konsekwencją rozwijania samych środowisk programistycznych, w tym wzbogacania ich o nowe funkcje, jest możliwość bezpośredniego wykorzystania ich w innych programach. To bardzo ważny kierunek rozwoju programowania wizualnego. W efekcie od pewnego czasu daje się zaobserwować coraz ściślejszą i szerszą integrację środowisk programowania parametrycznego (Dynamo, Grasshopper) z programami czy wręcz platformami do modelowania i analiz obliczeniowych obiektów budowlanych, a tym samym ich konstrukcji. Analizując ewolucję środowiska Dynamo, które od wielu lat jest dostępne jako samodzielny program, począwszy od jego wydania jako produktu firmy Autodesk pod nazwą Dynamo Studio 2017, następnie poprzez wprowadzenie go na stałe do głównego środowiska BIM Revit i dodanie do programu Robot 2022, można wysnuć wniosek, że kierunek w tym zakresie został już dawno wytyczony. Wraz z ciągłym rozwojem, coraz większym upowszechnieniem technologii BIM poszukiwane są narzędzia pozwalające na coraz większą automatyzację procesu nie tylko projektowania, ale tak naprawdę obsługi całego procesu realizacji inwestycji. Stąd m.in. wprowadzenie programowania wizualnego jako dodatku do centralnego programu zarządzania informacją BIM Revit.

Implementacja programowania w innych środowiskach ułatwia sam proces projektowania na wielu płaszczyznach, nie tylko modelowania obiektów. Inżynier nie musi już eksportować wyników swojej pracy w pośredni sposób np. za pomocą formatów uniwersalnych, bo efekt działania skryptu automatycznie jest już obecny

w danym modelu. Oczywiście sam proces tworzenia skryptu jest taki sam jak w przypadku aplikacji samodzielnej i wymaga od inżyniera takich samych umiejętności programowania. Jednakże samo już włączenie danego języka programowania wizualnego do środowiska projektowego ułatwia i skraca pracę, a jednocześnie zachęca osoby, które do tej pory nie miały z nim styczności do podjęcia próby stworzenia skryptu wizualnego.

Osobnym kierunkiem w omawianym zakresie jest obszar integracji środowisk programistycznych ze środowiskami projektowymi. W tym momencie wydaje się, że najbardziej efektywne jest jednak modelowanie geometrii i samo nią zarządzanie. Podstawowy jest ten drugi aspekt, biorąc pod uwagę samą koncepcję technologii BIM, gdzie zarządzanie informacją o modelu to sprawa fundamentalna. Operacje, jakie można przeprowadzać za pomocą skryptów wizualnych, dają bardzo duże możliwości w zakresie edycji modelu BIM, jego rozwijania czy porządkowania. Tym samym przyspieszony i ułatwiony jest proces automatyzacji modelowania BIM. Projektant ma możliwość przeprowadzania operacji na danych zawartych w samym modelu, jak również danych zewnętrznych. Ta druga opcja ma duże zastosowanie w realizacji choćby obiektów inżynierskich, które mogą być modelowane i lokalizowane w oparciu o rzeczywiste współrzędne punktów charakterystycznych inwentaryzowanych z natury. Osobną kategorią tego typu operacji jest praca na chmurze punktów pobieranych przy wykorzystaniu techniki skaningu 3D, która zaczyna stawać się podstawową metodą prowadzenia choćby inwentaryzacji obiektów budowlanych. Operacje bazodanowe, jakie mogą być przeprowadzane w oparciu o skrypt wizualny, są o wiele szersze niż możliwości samego środowiska modelowania.

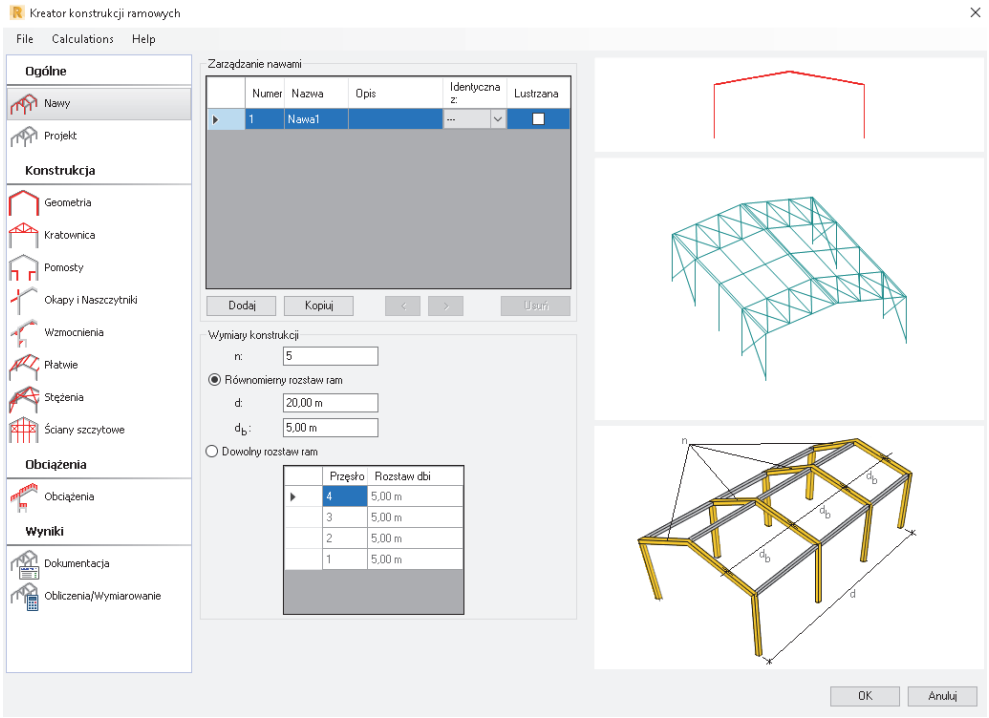
W zakresie integracji programowania wizualnego z innymi środowiskami bardzo istotnym elementem są również pakiety zawierające opcje programistyczne, które są przygotowywane dokładnie z określonym zakresem, co opisano już wcześniej. To istotny kierunek i element rozwoju środowisk programistycznych, bez którego trudno sobie dzisiaj wyobrazić uproszczenie pracy projektanta.

7.4. AUTOMATYZACJA PROJEKTOWANIA

Naturalną korzyścią stosowania w projektowaniu inżynierskim jakiegokolwiek oprogramowania jest większa lub mniejsza automatyzacja tego procesu. To samo odnosi się do programowania wizualnego, które umiejętnie stosowane potrafi zwiększyć wydajność w tym zakresie w stopniu nieosiągalnym, choćby w porównaniu do systemów CAD czy nawet BIM. Tym samym dalszy naturalny kierunek rozwoju tego środowiska to ewolucja w zakresie parametryzacji całego procesu projektowania obiektu, a nawet realizacji całej inwestycji. W rozdziale 5 pokazano kilka przykładów zastosowania skryptów do modelowania konstrukcji sparametryzowanych, które zaprezentowane od najprostszych do coraz bardziej skomplikowanych struktur ukazały niejako ewolucję tego procesu – parametryzacji prowadzącej do automatyzacji. Wydaje się, że w tym zakresie zastosowanie programowania wizualnego stwarza największe możliwości zastosowania. Właśnie kształ-

towanie skomplikowanych niesymetrycznych i niemodułowych geometrii jest jednym z największych problemów projektanta. Światowe trendy architektoniczne podążają od dłuższego już czasu w kierunku kształtowania form organicznych, bionicznych, czego przykładem mogą być choćby struktury rozmaitych obiektów budowlanych takich jak budynki czy mosty projektowane przez przywoływanych już architektów, takich jak Santiago Calatrava czy Zaha Hadid. Wydaje się, że właśnie w tym zakresie powinna nastąpić intensyfikacja działań na polu parametryzacji coraz to większej liczby konstrukcji niejako już typowych.

Zastosowanie zatem parametryzacji w samym procesie projektowania to jeden z podstawowych kierunków, w jakim powinien bardziej intensywnie rozwinąć się nurt programowania wizualnego. Choć kierunek ten został wytyczony całkiem dawno w zakresie programów, a obecnie aplikacji, niekoniecznie przewidzianych na komputery, to w zakresie programowania wizualnego wydaje się, że powinien zostać znacząco rozwinięty i zorientowany.



Rys. 7.1. Kreator konstrukcji ramowych w Robot Structural Analysis Professional

Analogicznie jak w przypadku projektowania generatywnego można by wzbogacić istniejące środowiska o pewne opcje, podzielone tematycznie, np. dla projektowania zadaszeń danego typu, obiektów inżynierskich, np. mostów czy typowych konstrukcji, np. hal o konstrukcji stalowej. Niejako prototypem mogłaby być opcja, która funkcjonuje w programie Robot pt. *Kreator konstrukcji ramowych*, a która

przewidziana jest do projektowania konstrukcji halowych. Użytkownik ma możliwość wygenerowania geometrii typowej hali o konstrukcji stalowej, może dokonać automatycznego obciążenia i weryfikacji w zakresie sprawdzenia i doboru przekrojów poszczególnych elementów nośnych (rys. 7.1).

Wzbogacenie istniejących środowisk programistycznych o tego typu funkcje na pewno znacznie ułatwiłoby i zautomatyzowałoby pracę inżyniera.

7.5. WYKORZYSTANIE SZTUCZNEJ INTELIGENCJI W INŻYNIERSKICH ŚRODOWISKACH PROGRAMISTYCZNYCH

W ostatnich kilku latach można zaobserwować intensyfikację działań poszczególnych producentów oprogramowań, a szerzej firm projektujących technologie i produkty IT, w zakresie rozwijania i upowszechniania technologii zwanej potocznie sztuczną inteligencją, czyli *AI* (ang. *Artificial Intelligence*). Tak na prawdę korzysta się z niej w życiu codziennym, często nawet nie zdając sobie z tego sprawy. Odblokowywanie smartfonu za pomocą odcisku palca czy wykorzystanie skaningu twarzy lub siatkówki oka promowane jako unikatowe sposoby zabezpieczania urządzeń wydają się być niedoskonałe, bo dość szybko opracowano aplikację opartą na AI, która potrafi wygenerować tzw. uniwersalny odcisk palca. W perspektywie kilku najbliższych lat AI będzie coraz agresywniej ingerować w nasze życie, próbując w jak największym stopniu odciążyć od wielu czynności wymagających od nas mniejszego bądź większego wysiłku intelektualnego.

Dlaczego więc nie wykorzystać AI w projektowaniu? Prace w tym zakresie trwają od dłuższego już czasu, a efektem ich jest choćby opracowanie narzędzi, które pozwalają inżynierowi na projektowanie generatywne. W tym przypadku AI nie odciąża nas od wysiłku, ale tak naprawdę pozwala na wykonanie pracy niemożliwej do wykonania w sposób manualny. Tak jak nie sposób mierzyć się z możliwościami chmury obliczeniowej, posiadając choćby nieźle wyposażaną jednostkę obliczeniową, tak nie ma sensu porównywania mocy obliczeniowej w zakresie możliwości generacji wariantów sytuacji, które można uzyskać przy wykorzystaniu projektowania generatywnego. Idąc dalej, dlaczego nie można pokusić się o próbę opracowania inteligentnych narzędzi programistycznych, które w najmniejszym choćby stopniu optymalizowałyby proces projektowy. Zastosowanie tutaj mogłyby znaleźć narzędzia używane już od jakiegoś czasu, jak choćby algorytmy genetyczne stosowane w optymalizacji różnych struktur konstrukcyjnych, np. [109, 110], jak i innych [111]. Przy narzuceniu odpowiednich kryteriów inżynier uzyskiwałby dostosowany do jego potrzeb produkt. Wydaje się, że to zbyt duży skok, jeśli chodzi o rozwój niezbyt przecież zaawansowanych narzędzi programistycznych stosowanych obecnie. Pogląd ten jest pozorny, bo tak naprawdę to tylko kwestia oprogramowania istniejącego już środowiska. Wymaga to oczywiście znacznego nakładu pracy, ale biorąc pod uwagę możliwości, jakie za sobą niesie, prędzej czy później powinno to nastąpić.

Możliwości, jakie od jakiegoś czasu są udostępniane przez producentów oprogramowania, pozwalają mieć nadzieje, że przez udoskonalanie oferowanych narzędzi i powiększanie obszarów ich zastosowań będzie można stosować tę technikę w codziennych aplikacjach, takich jak choćby optymalizacja masy projektowanej konstrukcji.

7.6. INNE DALSZE KIERUNKI ROZWOJU

Przyszłość pokaże, które z opisanych w monografii kierunków rozwoju programowania wizualnego w projektowaniu inżynierskim rozwiną się najszybciej i najintensywniej. Rozwój ten oczywiście będzie różny, w zależności od naturalnych potrzeb branży projektowej. Pewne ścieżki rozwoju doprowadzą zapewne do zaniechania działań i zmiany koncepcji stosowanych do tej pory. Należy jednakże mieć nadzieję, że projektowanie wizualne jest na krzywej wznoszącej, bo podstawowym, naturalnym niejako kierunkiem rozwoju jest automatyzacja i optymalizacja procesu projektowania. Istotnym elementem jest również kwestia nośnika dokumentacji technicznej, która prędzej czy później będzie tworzona w formie cyfrowej, będącej formą podstawową. Cyfrowy model obiektu budowlanego, który choćby w technologii BIM staje się jedynym kompletnym źródłem informacji, w perspektywie będzie więc jedynym, pełnym obrazem i nośnikiem zawierającym dokumentację techniczną i nie tylko. W tym zakresie programowanie wizualne dostarcza i będzie dostarczać coraz to nowych możliwości, pozwalając nie tylko na generację coraz to bardziej skomplikowanych geometrii, ale da niesamowite możliwości w zarządzaniu informacją zakodowaną w modelu.

Bibliografia

[1] Marczyński R., *Elektronowe automatyczne maszyny cyfrowe*, „Zastosowania Matematyki" 1954, R. 2, nr 3, s. 263-296.

[2] Haigh T., Priestley M., Rope C., *Engineering "The Miracle of the ENIAC": Implementing the Modern Code Paradigm*, "IEEE Annals of the History of Computing" 2014, Vol. 36, No. 2, s. 41-59.

[3] Randell B., *The COLOSSUS* [in:] *A History of Computing in the Twentieth Century*, Metropolis N., Hoelett J., Rota G.-C. (eds.), 1st ed., Academic Press, New York–London 1980, s. 47-92.

[4] Edwards J., *John Atanasoff and Clifford Berry: Inventing The ABC, A Benchmark Digital Computer*, Vol. 58, 2010, s. 58-59.

[5] Metropolis N., Hoelett J., Rota G.-C. (eds.), *A History of Computing in the Twentieth Century*, 1st ed., Academic Press, New York–London 1980.

[6] Maziarz P., *Pierwszy komputer obchodził 75. urodziny – zobacz jak rozwijały się pecety*, 2020, online: https://www.benchmark.pl/aktualnosci/historia-rozwoju-komputerow-i-laptopow.html#noop.

[7] Mahoney M.S., *The History of Computing in the History of Technology*, "Annals of the History of Computing" 1988, Vol. 10, No. 2, s. 113-125.

[8] Garland H., *Design Innovations in Personal Computers*, "Computer" 1977, Vol. 10, No. 3, s. 24-27.

[9] *The First Personal Computer*, "Popular Mechanic" 2000, s. 52.

[10] *Timeline of Computer History*, online: https://www.computerhistory. org/timeline/1971/#169ebbe2ad45559efbc6eb35720a58a7.

[11] Weisberg D.E., *The Engineering Design Revolution: The People, Companies and Computer Systems That Changed Forever the Practice of Engineering*, 2008.

[12] Sutherland I.E., *Sketchpad: A man-machine graphical communication system*, University of Cambridge, Cambridge 2003.

[13] Myers B.A., *A Brief History of Human Computer Interaction Technology*, "ACM Interactions" 1998, Vol. 5, No. 2, s. 44-54.

[14] Yares E., *50 Years of CAD*, "Design World" 2013, s. 66-71.

[15] Hamou N., *The keyboard and mouse are dead. Long live touch interactions*, 2015.

[16] Kossakowski P., *Zastosowanie technologii przetwarzania w chmurze obliczeniowej w procesie realizacji inwestycji budowlanych*, „Przegląd Budowlany" 2013, R. 84, nr 12, s. 62-67.

[17] Kossakowski P., *Cloud computing in engineering design*, „Przegląd Mechaniczny" 2014, R. 73, nr 2, s. 42-47.

[18] Grzebałkowska M., *Beksińscy. Portret podwójny*, 1 wyd., Kraków 2014.

[19] Eastman Ch., *The Use of Computers Instead of Drawings in Building Design*, "AIA Journal" 1975, Vol. 63, No. 3, s. 46-50.

[20] Eastman Ch., Teicholz P., Sacks R., Liston L., *BIM Handbook: A Guide to Building Information Modeling for Owners, Managers, Designers, Engineers, and Contractors*, 1st ed., John Wiley and Sons, Hoboken, New Jersey 2008.

[21] Aish R., *Building Modelling: The Key to Integrated Construction CAD*, The Fifth International Symposium on the Use of Computers for Environmental Engineering Related to Buildings, Guildhall, 7-9 July Bath 1986.

[22] van Nederveen G., Tolman F., *Modelling multiple views on buildings*, "Automation in Construction" 1992, Vol. 1, No. 3, s. 215-224.

[23] Laiserin J., *Comparing Pommes and Naranjas*, The Laiserin Letter, Vol. 15, 2002.

[24] Laiserin J., *LaiserinLetterLetters*, The Laiserin Letter, 6 January 2003.

[25] BS 1192:2007 Collaborative production of architectural, engineering and construction information. Code of practice.

[26] PAS 1192-2:2013 Specification for information management for the capital/delivery phase of construction projects using building information modelling.

[27] PAS 1192-3:2014 Specification for information management for the operational phase of assets using building information modelling.

[28] BS 1192-4:2014 Collaborative production of information – Part 4: Fulfilling employer's information exchange requirements using COBie – Code of practice.

[29] PAS 1192-5:2015 Specification for security-minded building information modelling, digital built environments and smart asset management.

[30] PD 19650-0:2019 Transition guidance to BS EN ISO 19650.

[31] ISO 19650-1:2018 Organization and digitization of information about buildings and civil engineering works, including building information modelling (BIM) – Information management using building information modelling – Part 1: Concepts and principles.

[32] ISO 19650-2:2018. Organization and digitization of information about buildings and civil engineering works, including building information modelling (BIM) – Information management using building information modelling – Part 2: Delivery phase of the assets.

[33] ISO 19650-3:2020 Organization and digitization of information about buildings and civil engineering works, including building information modelling (BIM) – Information management using building information modelling – Part 3: Operational phase of the assets.

[34] ISO/DIS 19650-4 Organization and digitization of information about buildings and civil engineering works, including building information modelling (BIM) – Information management using building information modelling – Part 4: Information exchange.

[35] ISO 19650-5:2020 Organization and digitization of information about buildings and civil engineering works, including building information modelling (BIM) – Information management using building information modelling – Part 5: Security-minded approach to information management.

[36] Sacks R., Eastman C., Lee G., Teicholz P., *BIM Handbook: A Guide to Building Information Modeling for Owners, Designers, Engineers, Contractors, and Facility Managers*, John Wiley and Sons, 3rd ed., Hoboken, New Jersey 2018.

[37] Tomana A., *BIM – Innowacyjna technologia w budownictwie. Podstawy, standardy, narzędzia*, PWB Media, Warszawa 2016.

[38] Kasznia D., Magiera J., Wierzowiecki P., *BIM w praktyce: standardy wdrożenia case study*, Wydawnictwo Naukowe PWN, Warszawa 2017.

[39] *The 7 dimensions of BIM – 3D, 4D, 5D, 6D, 7D BIM explained*, ACCA software, 2021, online: https://biblus.accasoftware.com/en/bim-dimensions/.

[40] *Building Information Modeling: The future of construction*, 2019, online: https://www.letsbuild.com/blog/bim-maturity-levels

[41] Kossakowski P., *Modelowanie informacji o budynku (BIM) – obowiązkowy standard przyszłości?*, „Przegląd Budowlany" 2014, R. 84, nr 4, s. 48-50.

[42] Kossakowski P., *Zastosowanie systemu obiektowej informacji o konstrukcji w projektowaniu CAD*, "Systems. Journal of Transdisciplinary Systems Science" 2012, Vol. 16, No. 1, s. 279-288.

[43] Jędrych K., Poślada J., Latała M., Zieliński T., *OpenRoads Designer. Projektowanie dróg w BIM*, Multiconsult Polska, 2020.

[44] Helenowska-Peschke M., *Parametryczno-algorytmiczne projektowanie architektury*, Wydawnictwo Politechniki Gdańskiej, Gdańsk 2014.

[45] Novak M., *Liquid Architectures in Cyberspace* [in:] *Cyberspace: first steps*, Benedikt M. (ed.), MIT Press, Cambridge 1991, s. 225-254.

[46] Kolarevic B., *Digital Morphogenesis* [in:] *Architecture in the Digital Age. Design and Manufacturing*, Kolarevic B. (ed.), 1st ed., Taylor and Francis, London 2003, s. 29.

[47] Tomoko S., Ferré A., *From Control to Design*: *Parametric/algorithmic Architecture*, Actar-D, Barcelona 2008.

[48] Woodbury R., *Elements of Parametric Design*, Routledge 2010.

[49] Burry J., Burry M., *The New Mathematics of Architecture*, Thames and Hudson, London 2010.

[50] Burry M., *Scripting Cultures: Architectural Design and Programming*, Wiley, Chichester 2011.

[51] Wassim J., *Parametric Designfor Architecture*, Laurence King, 2013.

[52] Caneparo L., *Digital Fabrication in Architecture, Engineering and Construction*, Springer, Dordrecht 2013.

[53] Brady P., *Inside Smartgeometry: Expanding the Architectural Possibilities of Computational Design*, Wiley, Chichester 2013.

[54] Steenson Wright M., *Architectural Intelligence*, MIT Press, Cambridge 2017.

[55] Cogdell C., *Toward a Living Architecture?: Complexism and Biology in Generative Design*, University of Minnesota Press, 2018.

[56] Bonenberg W., Giedrowicz M., Radziszewski K., *Współczesne projektowanie parametryczne w architekturze*, Wydawnictwo Politechniki Poznańskiej, Poznań 2019.

[57] Witruwiusz, *O architekturze ksiąg dziesięć*, Prószyński i S-ka, 1999.

[58] Strelau J., *Psychologia. Podręcznik akademicki – podstawy psychologii*, cz. 1, Gdańskie Wydawnictwo Psychologiczne, Gdańsk 2006.

[59] Diggins C., *MCG: Visual Functional Programming*, 2015, online: https://area.autodesk.com/blogs/the-3ds-max-blog/mcg-visual-functional-programming/.

[60] *Visual programming language*, 1995, online: https://foldoc.org/visual+programming+language.

[61] *Faqs.org*, 1998, online: http://www.faqs.org/faqs/visual-lang/faq/.

[62] *Wizualny język programowania – Visual programming language*, 2021, online: https://pl.abcdef.wiki/wiki/Visual_programming_language.

[63] Burnett M.M., Baker M.J., *A Classification System for Visual Programming Languages*, Corvallis 1994.

[64] Burnett M.M., Baker M.J., *A Classification System for Visual Programming Languages*, "Journal of Visual Languages and Computing" 1994, Vol. 5, No. 3, s. 287-300.

[65] Chang S.-K., Levialdi S., *Foreword*, "Journal of Visual Languages and Computing" 1990, Vol. 1, No. 1, s. 1, 2.

[66] *Journal of Computer Languages. About the journal – Aims and scope*, 2021, online: https://www.sciencedirect.com/journal/journal-of-computer-languages/about/aims-and-scope.

[67] Kowalczyk M., *Charakterystyka wizualnych, edukacyjnych języków programowania*, 2019, online: https://www.edukacja.edux.pl/p-41470-charakterystyka-wizualnych-edukacyjnych.php.

[68] https://www.appinventor.mit.edu.

[69] https://scratch.mit.edu.

[70] Kossakowski T., *Skrypt Scratch*, Kielce 2021.

[71] Szlagor P., *Programowanie wizualne dla każdego. Przewodnik po Scratch*, 1 wyd., Think Global, Warszawa 2013.

[72] *DEAFCODE. Podręcznik zawierający 20 scenariuszy zajęć, które można prowadzić z wykorzystaniem bezpłatnego wideo-kursu, zamieszczonego na stronie www.pzg.lodz.pl/deafcode*, Polski Związek Głuchych. Oddział Łódzki.

[73] *Grasshopper 3D*, 2021, online: https://pl.abcdef.wiki/wiki/Grasshopper_3d.

[74] *Features*, 2021, online: https://www.rhino3d.com/features/.

[75] Tedeschi A., *Parametric Architecture with Grasshopper*, Primer, Le Pensuer, 2011.

[76] Tedeschi A., *AAD Algorithms-Aided Design: Parametric Strategies using Grasshopper*, Le Pensuer, Brienza 2014.

[77] Khabazi Z., *Generative Algorithms using Grasshopper*, 3rd ed., Morphogenesism, January 2012.

[78] *DIG: Rhino models/scene setup for Architectural Illustrations*, 2016, online: https://discourse.mcneel.com/t/dig-rhino-models-scene-setup-for-architectural-illustrations/31434.

[79] *Highest #DynamoBIM Version for #Revit Versions*, 2017, online: https://www.sixtysecondrevit.com/2017-03-07-highest-dynamobim-version-for-revit-versions/.

[80] *Finding the best sound*, online: https://dynamobim.org/home_usecases/use-case-2/.

[81] *Marionette Tutorial – Part 4 – Object Nodes*, (film), Vectorworks, 2015.

[82] *Marionette Tutorials*, 2016, online: https://forum.vectorworks.net/index.php?/articles.html/articles/marionette-tutorials-r432.

[83] *Artistic Design With Marionette in Vectorworks*, (film).

[84] Tutorials, 2011, online: https://communities.bentley.com/products/products_generative components/w/generative_components_community_wiki/4917/tutorials.

[85] Feist S., *A-BIM: Algorithmic-based Building Information Modelling*, 2016.

[86] *Przewodnik Dynamo Primer*, 2019, online: https://primer.dynamobim.org/pl/.

[87] Zerbst R., *Gaudi. Wszystkie budowle*, Taschen, 2010.

[88] Carrillo de Albornoz Fisac C., *Santiago Calatrava (Ultimate Collection)*, Assouline, 2013.

[89] Jodido P., *Zaha Hadid. Complete Works 1979 – Today*, 2020.

[90] *Steel Structure Detail*, online: https://havitsteelstructure.com/steel-structure-detail/.

[91] *Offsite manufacturing in the frame*, 2020, online: http://www.newsteelconstruction.com/wp/offsite-manufacturing-in-the-frame/.

[92] https://catavino.net/, online: https://c2.staticflickr.com/6/5524/9757556574_0e6850b509_b.jpg.

[93] https://i.pinimg.com/originals/3e/2d/74/3e2d74f0d9dd200aff98f4a566302195.jpg.

[94] https://live.staticflickr.com/3228/2722727827_66d91dba95_b.jpg.

[95] *Dynamo Practice 12. The Calatrava Agora*, (film), DynaMorfine.

[96] https://i.pinimg.com/originals/d6/e7/ee/d6e7eea004b3355acc4a122c29180125.jpg.

[97] *2021 UEFA Europa League Final to be played on Hatko Hybridgrass*, online: https://www.hatkosport.com/2021-uefa-europa-league-final-to-be-played-on-hatko-hybridgrass/.

[98] *The Helix Bridge*, online: https://www.coxarchitecture.com.au/project/the-helix-bridge/.

[99] Krish S., *A practical generative design method*, "Computer-Aided Design" 2011, s. 88-100.

[100] Dapogny C., Faure A., Michailidis G., Allaire G., Couvelas A., Estevez R., *Geometric constraints for shape and topology optimization in architectural design*, "Computational Mechanics" 2017, Vol. 59, No. 6, s. 933-965.

[101] *Dynamo Development Builds*, online: https://dynamobuilds.com/.

[102] *This is Dynamo*, online: http://dynamobim.org/download/.

[103] *DynamoDS/Dynamo*, online: https://github.com/DynamoDS/Dynamo.

[104] *Learn – DynamoBIM*, online: https://dynamobim.org/.

[105] https://www.rhino3d.com/learn/?keyword=kind:%20grasshopper.

[106] https://www.rhino3d.com/learn/?keyword=kind:%20rhino_win.

[107] *Dynamo Language Manual*, online: https://dynamobim.org/wp-content/uploads/forum-assets/colin-mccroneautodesk-com/07/10/Dynamo_language_guide_version_1.pdf .

[108] *Dynamo: Visual Programming for Design*, online: https://help.autodesk.com/ sfdcarticles/attachments/Dynamo_Visual_Programming_for_Design.pdf.

[109] Toropov V., Mahfouz S., *Design optimization of structural steelwork using a genetic algorithm, FEM and a system of design rules*, "Engineering Computations" 2001, Vol. 18, No. 3/4, s. 437-460.

[110] Łodygowski T., Szajek K., Wierszycki M., *Optymalizacja konstrukcji półskorupowej z użyciem algorytmu genetycznego*, „Górnictwo Odkrywkowe" 2010, R. 51, nr 3, s. 88-92.

[111] Łodygowski T., Szajek K., Wierszycki M., *Optimization of dental implant using genetic algorithm*, "Journal of Theoretical and Applied Mechanics" 2009, Vol. 47, No. 3, s. 573-598.