Państwowa Wyższa Szkoła Zawodowa w Nowym Sączu

Sławomir Jurkowski

Grafika i komunikacja człowiek – komputer

Wprowadzenie do oprogramowania wspomagającego modelowanie, animację i rendering – Blender 3D (wersja 2,9), POV-Ray (wersja 3,7)

Nowy Sącz 2021

Redaktor Naukowy
mgr inż. Sławomir Jurkowski

Redaktor Wydania
prof. dr hab. inż. Adam Ruszaj

Recenzja
dr hab. Olesia Afanasieva, prof. UP

Redaktor Techniczny
dr Tamara Bolanowska-Bobrek

ISBN 978-83-65575-78-4

Wydawca
Wydawnictwo Naukowe Państwowej Wyższej Szkoły Zawodowej w Nowym Sączu
ul. Staszica 1, 33-300 Nowy Sącz
tel.: +48 18 443 45 45, e-mail: wn@pwsz-ns.edu.pl

Adres Redakcji
Nowy Sącz 33-300, ul. Staszica 1
tel.: +48 18 443 45 45, e-mail: tbolanowska@pwsz-ns.edu.pl

Druk
Wydawnictwo i drukarnia NOVA SANDEC s.c.
Mariusz Kałyniuk, Roman Kałyniuk
33-300 Nowy Sącz, ul. Lwowska 143
tel.: +48 18 547 45 45, e-mail: biuro@novasandec.pl

Spis treści

1. Wprowadzenie

Niniejszy skrypt został opracowany jako pomoc dla studentów kierunku Informatyka, którzy odbywają zajęcia z przedmiotu „Grafika i komunikacja człowiek – komputer", realizowanych w Instytucie Technicznym Państwowej Wyższej Szkoły Zawodowej w Nowym Sączu.

Podstawowym oprogramowaniem, na jakim oparty został przedmiot jest Blender 3D, a dodatkowo – w ramach rozwoju umiejętności programistycznych w kontekście grafiki komputerowej – wykorzystywany jest program POV-Ray. Realizacja projektów graficznych z wykorzystaniem przedstawionych oprogramowań ma na celu realizację w ramach przedmiotu efekty kształcenia:

- Student objaśnia teorię grafiki komputerowej.
- Student używa i analizuje podstawowe algorytmy grafiki komputerowej.
- Student planuje i tworzy realistyczne sceny 3D w aplikacjach: POV-Ray, Moray oraz Blender.
- Student dostrzega postęp naukowy i technologiczny oraz kreatywnie rozwija metody/ techniki grafiki i wizji komputerowej.

Na rynku dostępne są opracowania dotyczące Blendera w wersjach wcześniejszych niż 2.8. Zauważalny jest postęp w zakresie dostępnych narzędzi i funkcjonalności, od wersji 2.8, oraz całkowite przeorganizowanie wyglądu oprogramowania, zastosowanych silników renderujących oraz nazewnictwa i funkcjonalności narzędzi. W tym kontekście zasadne jest udostępnienie studentom praktycznego opracowania do aktualnej wersji programu – 2.9.

Opisywany jako drugi w skrypcie POV-Ray, poza stroną internetową twórców i pomocą wewnętrzną oprogramowania, nie posiada natomiast opisu książkowego. Przedstawione w tym opracowaniu materiały, dostosowane do wykorzystania oprogramowania wraz z językiem programowania SDL (Scene Description Language), zostały dostosowane pod kątem wykorzystania inżynierskiego.

2. Co to jest Blender?

Oprogramowanie Blender swoimi korzeniami sięga do roku 1988, w którym to Ron Roosendaal utworzył studio animacji „NeoGeo". Narzędzia wykorzystywane w nim zostały zebrane w jedno oprogramowanie – Blender 1.0 (1995).

Po konferencji Siggraph w 1999 roku notowano szybki rozwój oprogramowania łącznie z integracją w nim silnika do gier, jednak dopiero po założeniu Blender Fundation w 2002 roku rozpoczęto prace nad pierwszą „poważną" odsłoną Blendera w wersji 2.26. Została ona wypuszczona na rynek na licencji Open Source. Od tego czasu ewoluuje i jest dynamicznie rozwijana. Pozostaje jednak wciąż darmowym oprogramowaniem, co może dziwić z punktu widzenia jego ogromnych możliwości w porównaniu z komercyjnymi alternatywami na rynku oprogramowań graficznych.

Większość opracowań, jakie można spotkać na rynku wydawniczym, jak np. „Blender. Kompendium" (Kuklo, Kolmaga, 2007) lub „Blender 3D Basics Beginner's Guide" (Fisher, 2012), stanowi dobrą bazę do rozpoczęcia pracy z oprogramowaniem Blender 3D. Spotkać można również zaawansowane opracowania nastawione na tworzenie efektów, zbliżonych jakością do pełnometrażowych filmów i animacji kinowych. W tym przypadku „Mastering Blender" (Mullen, 2009) można uznać za „lekturę obowiązkową".

W każdym z przytoczonych przykładów pracowano na wersjach oprogramowania bardzo odbiegających od aktualnych odsłon Blendera. Co do koncepcji działań i logiki operacji jak najbardziej stanowią cenne źródło wiedzy. Polskojęzycznych publikacji opisujących najnowszą wersję programu 2.9 nie ma, zaś anglojęzyczne opracowania pojawiły się pod koniec 2020 roku, a ich dostępność jest utrudniona.

2.1 Możliwości

Do czego więc służy Blender? Można powiedzieć, że „do wszystkiego", te kolokwialne stwierdzenie może jest pewnym nadużyciem, lecz oprogramowanie posiada samo w sobie narzędzia pozwalające na szeroką gamę prac z różnych obszarów grafiki komputerowej, a dodatkowo istnieje wiele wtyczek rozszerzających jego możliwości. Podstawowe funkcjonalności Blendera to:

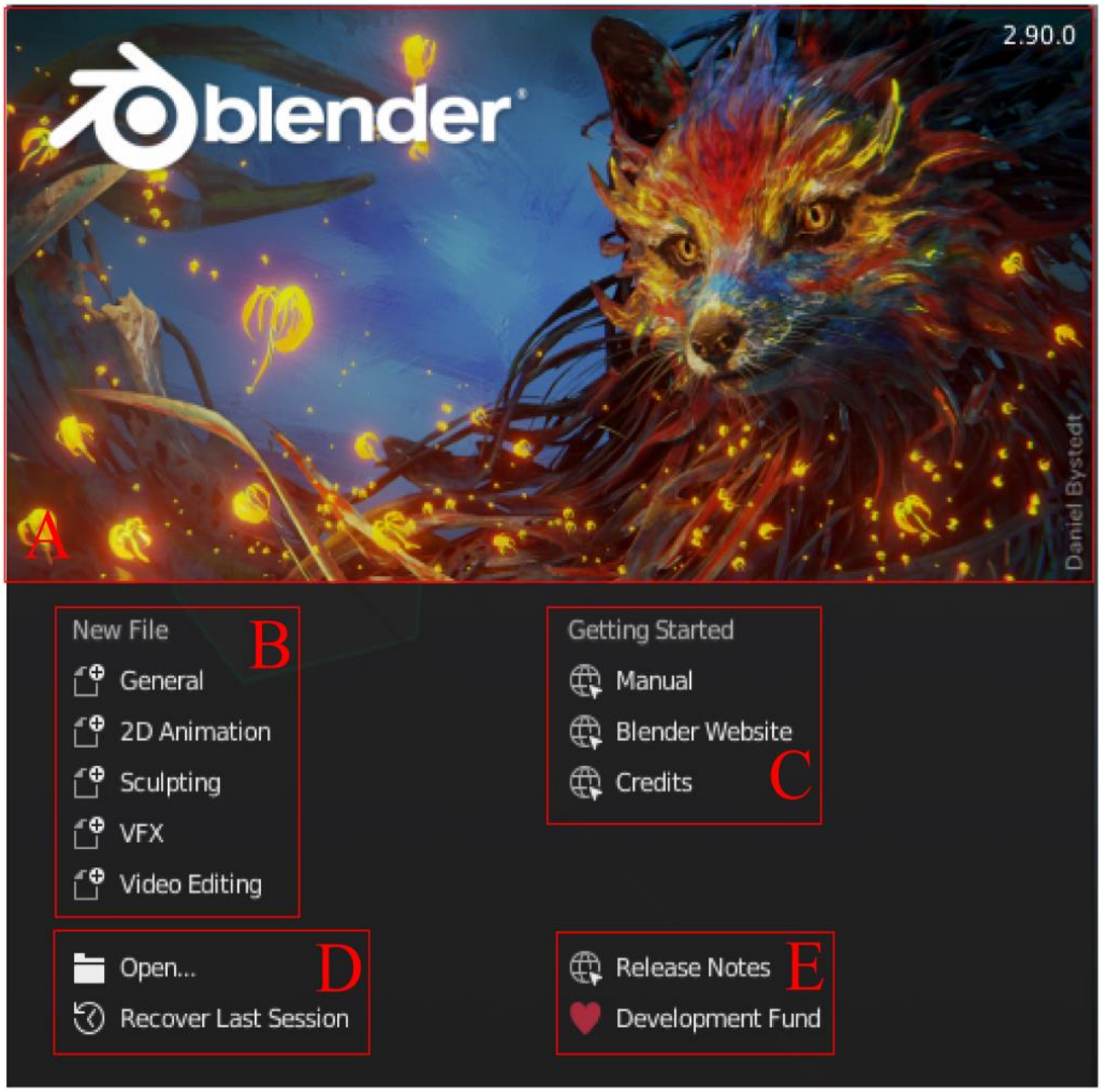
- Tworzenie statycznych wizualizacji 3D.
- Animacja 3D.
- Teksturowanie.
- Renderowanie.
- Symulacje.
- Generowanie filmowych efektów specjalnych.
- Programowania obrabiarek (mniej znane zastosowanie).

3. Interface – Blender 2.9

Opis układu okien i narzędzi dostępnych w oprogramowania Blender 3D ma na celu odnalezienie pożądanych funkcji oraz łatwą nawigację pomiędzy poszczególnymi elementami składowymi interface‘u.

3.1 Splash screan

Po uruchomieniu Blendera naszym oczom ukazuje się tzw. „Splash screen” – okno powitalne 3.1, czyli niejako wizytówka oprogramowania, która jest znakiem rozpoznawczym Blendera, ponieważ każda wersja okraszona jest autorską wizualizacją, pokazującą możliwości danej wersji. „Splash screen” jest również oknem szybkiej konfiguracji.

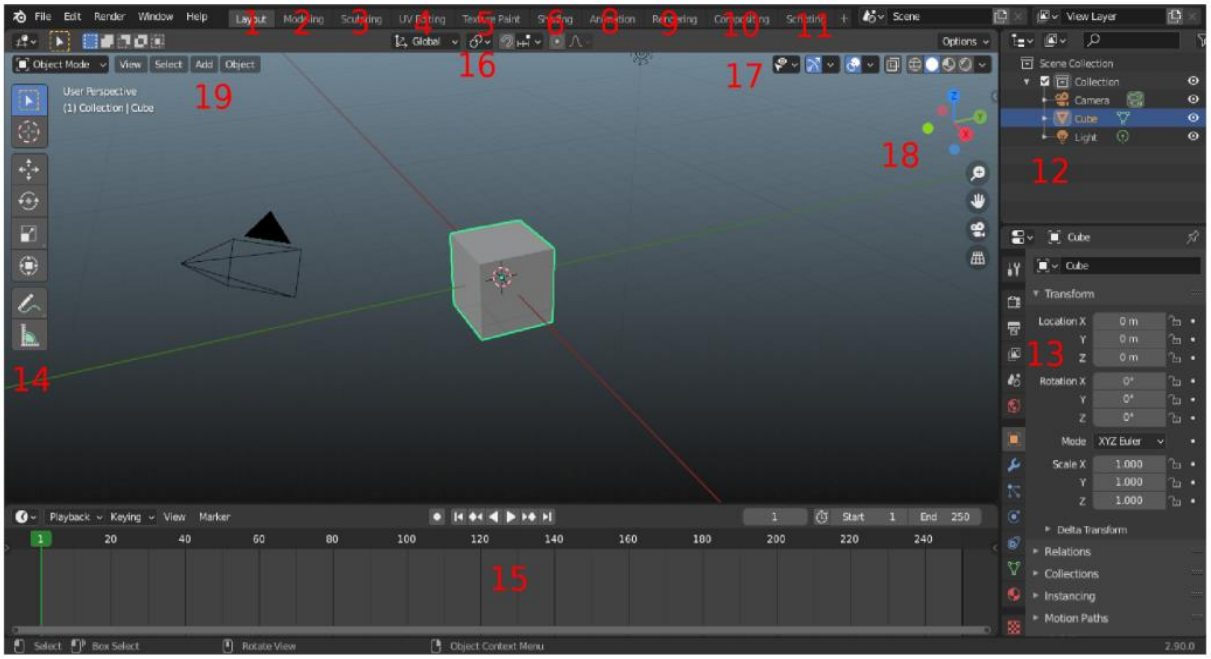


Rysunek 3.1. Okno powitalne.
Źródło: opracowanie własne.

Funkcje:

- Information Region – Górna część to wizualizacja wersji Blendera. Informacje o aktualnie otwartej wersji znajdziemy w prawym górnym rogu okna (A).
- Interactive Region – Dolna część „Splash screen" – jest to obszar szybkiej konfiguracji, źródeł informacji i pomocy:
    - New File – Rozpocznij nowy projekt korzystając z gotowego schematu (B).
    - Recent Files – Najczęściej otwierane pliki (C), przy pierwszym uruchomieniu w tej sekcji znajdują się linki do dokumentacji technicznej Blendera.
    - Open – Otwórz istniejący plik (D).
    - Recover Last Session – Odzyskaj ostatnią sesję.
    - Links – Hiperłącza do użytecznych stron

3.2 Okna i elementy menu



Rysunek 3.2. Okno główne
Źródło: opracowanie własne.

Tryby pracy oprogramowania najłatwiej wybrać korzystając z wstążki, na której znajdują się odpowiednie zakładki – okno 3.2 1-11. Wybór konkretnego trybu zmienia wygląd okien, włącza lub wyłącza odpowiednie funkcjonalności i przybornik narzędziowe. Opis wykorzystania poszczególnych zakładek przedstawiono poniżej:

- Layout Domyślne ustawienie układu okien, które jako pierwsze ma do dyspozycji użytkownik 3.2-1. Podstawowe obszary to okno podglądu 3D w centrum, Outliner – drzewo struktury scenerii 3.2-12, panel ustawień 3.2-13 oraz oś czasu 3.2-15. Przybornik narzędziowy na ogół niezależnie od trybu pracy znajduje się w lewej części obszaru okna 3.2-14.
- Modeling – okno z powiększonym obszarem podglądu 3D 3.2-2, nie ma wyświetlonej tutaj osi czasu.
- Sculpting – moduł rzeźbiarski, umożliwiający skorzystanie z narzędzi do modyfikacji geometrii, polegających na swobodnym deformowaniu siatki obiektów 3.2-3.

- UV Editing – zestaw narzędzi do rozwijania siatki obiektów modyfikacji oraz nakładania tekstur 3.2-4.
- Texture Painting – narzędzia do malowania powierzchni obiektów, wykorzystywane również m.in. w generowaniu systemów cząsteczkowych 3.2-5.
- Shading – obszar nadawania cech materiałowych obiektom oraz otoczeniu 3.2-6, poprzez zastosowanie zaawansowanych narzędzi i modyfikatorów – tzw. Nodes.
- Animation – narzędzia do ustawienia animacji 3.2-8 (dodatkowo analogicznie występuje tryb 2D Animation w przypadku animowania scenerii dwuwymiarowych).
- Rendering – odtworzenie scenerii z wykorzystaniem silnika graficznego oprogramowania (domyślnie jeden z trzech silników renderujących: Cycles, Eevee, Workbench), podgląd i analiza wyników obliczeń scenerii 3.2-9.
- Compositing – narzędzia do post-procesingu wyników renderingu 3.2-10.
- Scripting – narzędzie umożliwiające wykorzystanie języka Python w celu automatyzacji pracy nad scenerią oraz zaawansowanych narzędzi w niemal wszystkich obszarach funkcjonowania oprogramowania 3.2-11.

4. Skróty klawiszowe

Użytkowanie oprogramowania Blender zasadniczo nie wymaga znajomości skrótów klawiszowych, służących do obsługi poszczególnych funkcji, narzędzi itp. Wszystkie funkcje można odnaleźć w menu czy w zestawach przycisków wyświetlanym na ekranie w zależności od aktualnego trybu, w jakim znajduje się oprogramowanie.

Z punktu widzenia wydajności pracy nad projektem jednak najlepsza strategia to korzystanie jedną ręką z myszki, a drugą operowanie na klawiaturze, wybierając kolejne polecenia. Nie tracimy wtedy czasu na „namierzanie" przycisków czy rozwijanie kolejnych zakładek menu.

W rozdziale przedstawiono najbardziej przydatne skróty klawiszowe wspomagające pracę z oprogramowaniem.

4.1 Ogólne

Przemieszczanie obiektów ułatwia korzystanie z następujących zestawów skrótów:

- **G** Przesuń
- **S** Skaluj
- **R** Obróć

Po wybraniu R, S, G istnieje możliwość operacji realizowanej po jednej z osi, wciskając na klawiaturze X, Y lub Z. Podwójne kliknięcie litery oznacza przekształcenie po wybranej osi, ale w lokalnym układzie współrzędnych.

Podstawowe skróty funkcjonujące w niemal wszystkich typach okien (trybów pracy):

- **T** Ukrycie lub odkrycie palety narzędziowej
- **N** Ustawienia obiektu, narzędzi, widoku
- **Shift + A** Dodaj obiekt lub węzła
- **X** lub **Delete** Usuwanie
- **F3** Wyszukiwarka funkcji
- **Rx2** Obrót przestrzenny
- **Shift** Dokładne przesunięcie (obrót, skalowanie; przytrzymanie klawisza zmienia reakcję między ruchem myszki, a reakcją obiektu podczas transformacji)
- **Ctrl** Przyciąganie do siatki (Przytrzymanie klawisza umożliwia skokowe przemieszczanie obiektu, przybliżenie/oddalenie widoku zmienia skalowanie)
- **Shift + D** Kopiowanie
- **Alt + D** kopiowanie połączone
- **H** Ukryj
- **Alt + H** Odkryj wszystko
- **Shift + H** Ukryj wszystko poza zaznaczonym
- **D + LPM** Komentarze odręczne (Przytrzymany klawisz) + lewy przycisk myszy (LPM)
- **D + PPM** Wymarz komentarz odręczny (Przytrzymany klawisz) + prawy przycisk myszy (PPM)
- **Q** Menu ulubionych poleceń

4.2 Nawigacja w obszarze roboczym

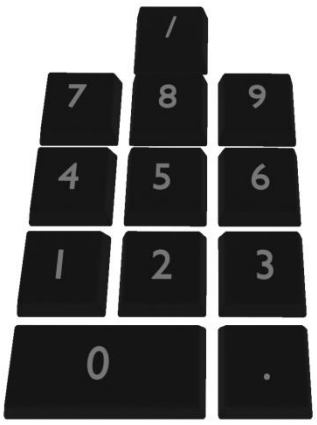
Swobodne poruszanie się po scenie jest możliwe przy wykorzystaniu myszki:

- **ŚPM** Orbita (obrót)
- **Shift + ŚPM** Rączka (przesuwanie widoku)
- **Ctrl + ŚPM** Przybliżanie lub oddalanie (alternatywą jest obrót kółkiem myszki)
- **Shift + ˜** Tryb lotu (sterowanie: W, S, A, D)

Najczęściej stosowaną metodą zmiany widoku w obszarze roboczym jest skorzystanie z klawiatury numerycznej. Każdej cyfrze przypisane są określone rzuty lub przekształcenia 4.1

Funkcjonalność poszczególnych klawiszy to:

- **1** Rzut z przodu
- **3** Rzut z boku
- **7** Rzut z góry
- **9** Rzut z tyłu
- **2** Obrót w dół
- **8** Obrót w górę
- **4** Obrót w lewo
- **6** Obrót w prawo
- **5** Włączenie/wyłączenie perspektywy
- **0** Widok z kamery



Rysunek 4.1. Sterowanie widokami.
Źródło: opracowanie własne.

- **.** Zbliżenie
- **/** Wyizolowanie widoku na zaznaczony obiekt
- Dodatkowo nawigację ułatwiają:
- **˜** Menu radarowe z rzutami
- **Alt + ŚPM** ”Przeciągnięcie” widoku do sąsiedniego rzutu
- **Home** Pokazanie wszystkich obiektów
- **Shift + B** Przybliżenie zaznaczonego obszaru

4.3 Tryb obiektowy

- **Ctrl + TAB** Menu radarowe z trybami pracy
- **TAB** Przełączanie między trybem obiektowym, a trybem edycji
- **Ctrl + M** Lustro (X, Y, Z wybieramy po jakiej osi ma być odbicie lustrzane)
- **Ctrl + P** Ustawienie zależności „Rodzic – dziecko" – jeden obiekt śledzi drugi
- **Alt + P** Usunięcie zależności „Rodzic – dziecko" – jeden obiekt śledzi drugi
- **Shift + TAB** Zmiana przyciągania elementów
- **Alt + G** Wyczyszczenie transformacji
- **Alt + R** Wyczyszczenie rotacji
- **Alt + S** Wyczyszczenie skalowania
- **Ctrl + A** Zatwierdzenie translacji, rotacji lub skalowania
- **Ctrl + J** połączenie wybranych obiektów
- **Ctrl + L** Skopiowanie atrybutów do nowego obiektu
- **Ctrl + 0/1/2/3/4/5** Dodanie poziomu podziału bryły (zmiana ilości węzłów)
- **Alt + B** Dodanie/usunięcie maski do obszaru
- **Shift + C** Wyśrodkowanie na kursor 3D
- **M** Przeniesienie aktywnego obiektu do kolekcji
- **Ctrl + Alt + 0** Przeniesienie kamery do aktualnego widoku
- **Ctrl + 0** Aktywacja kamery

Przydatne w przypadku manipulacji obiektami w oknie podglądu są też dwa polecenia:

- **Alt + Z** Włączenie i wyłączenie Promieni-X (przejrzystość obiektów ułatwiająca edycje)
- **Shift + S** Wywołanie menu kontekstowego manipulacji z wykorzystaniem kursora 3D

4.4 Metody zaznaczania/wybierania elementów

W większości przypadków można zastosować metody ułatwienia zaznaczanie elementów i obiektów, zwalniając użytkownika z konieczności ręcznego „klikania" w każdy obiekt oddzielnie w celu jego wybrania/odwołania wyboru.

- **LPM** Wybierz/zaznacz
- **A** Zaznacz wszystko
- **Alt + A** lub **Ax2** Odznacz wszystko
- **B** Wybór zaznaczeniem prostokątnym
- **C** Wybór zaznaczeniem okrągłym (obrót kółkiem myszy zmienia promień zaznaczenia)
- **Ctrl + PPM** Zaznaczeniem lassem
- **Ctrl + i** Odwrócenia zaznaczenia
- **Shift + L** Zaznacz połączone
- **Shift + G** Zaznacz podobne
- **Alt + LPM** Wywołanie listy obiektów wokół kursora możliwych do zaznaczenia

W szczególności opcje działające tylko w trybie edycji:

- **Ctrl + L** Zaznacz połączone elementy
- **L** Zaznacz połączone elementy pod kursorem
- **Alt + LPM** Zaznacz pętlę krawędzi

- **Ctrl + Alt + PPM** Zaznacz pierścień
- **1** Tryb wyboru punktów
- **2** Tryb wyboru krawędzi
- **3** Tryb wyboru powierzchni
- **Ctrl + Shift + M** Odbicie lustrzane zaznaczenia
- **Ctrl +/-** Rozszerz/pomniejsz zaznaczenie
- **Ctrl + E** Menu operacji na krawędziach

4.5 Edycja podczas modelowania

Modyfikacje krzywych realizuje się z wykorzystaniem następujących poleceń:

- **E** lub **Ctrl + PPM** Dodawanie punktów
- **V** Zmiana typy uchwytów modyfikujących
- **Ctrl + X** Usuń, pozostawiając połączenie
- **Alt + C** Zamknij krzywą
- **Ctrl + T** Pochylenie
- **Alt + T** Usuń pochylenie

  Skróty przy modelowaniu przestrzennym:
- **E** Wydłuż
- **i** Odsunięcie krawędzi (wstawia powierzchnię w powierzchni)
- **Ctrl + B** Fazowanie krawędzi
- **Ctrl + R** Natnij w koło
- **Gx2** Przesuń, ślizgając po krawędzi
- **K** Nóż
- **F** Wypełnij powierzchnię
- **Ctrl + Shift + Alt + S** Przekręć
- **Shift + W** Wygnij
- **Y** Oddziel
- **V** Oderwanie punktu, krawędzi lub płaszczyzny
- **Alt + V** Rozerwanie z wypełnieniem
- **M** Scal zaznaczone
- **Shift + N** Przelicz wektory normalne
- **Ctrl + Shift + N** Odwróć wektory normalne

  Edycja proporcjonalna
- **Shift + O** Menu radarowe opcji edycji proporcjonalnej
- **P** Oddziel, tworząc nowy obiekt

4.6 Node Editor

Skróty pomagające w edycji i manipulacji węzłami w ”Node Editor”:

- **Ctrl + PPM** Przetnij połączenie
- **F** Połącz zaznaczone
- **N** Wyświetl panel ustawień
- **Ctrl + X** Usuń, pozostawiając połączenia
- **Ctrl + Shift + D** Skopiuj zaznaczone, utrzymując połączenia
- **M** Wygaś
- **Ctrl + G** Zgrupuj
- **Ctrl + Alt + G** Rozdziel grupę
- **Tab** Przełącz grupę
- **Ctrl + J** Utwórz ramkę
- **Ctrl + H** Ukryj/pokaż nieużywane piny

4.7 Tryb rzeźbienia

Skróty pomagające w pracy przy swobodnym modelowaniu w module „Sculpting”:

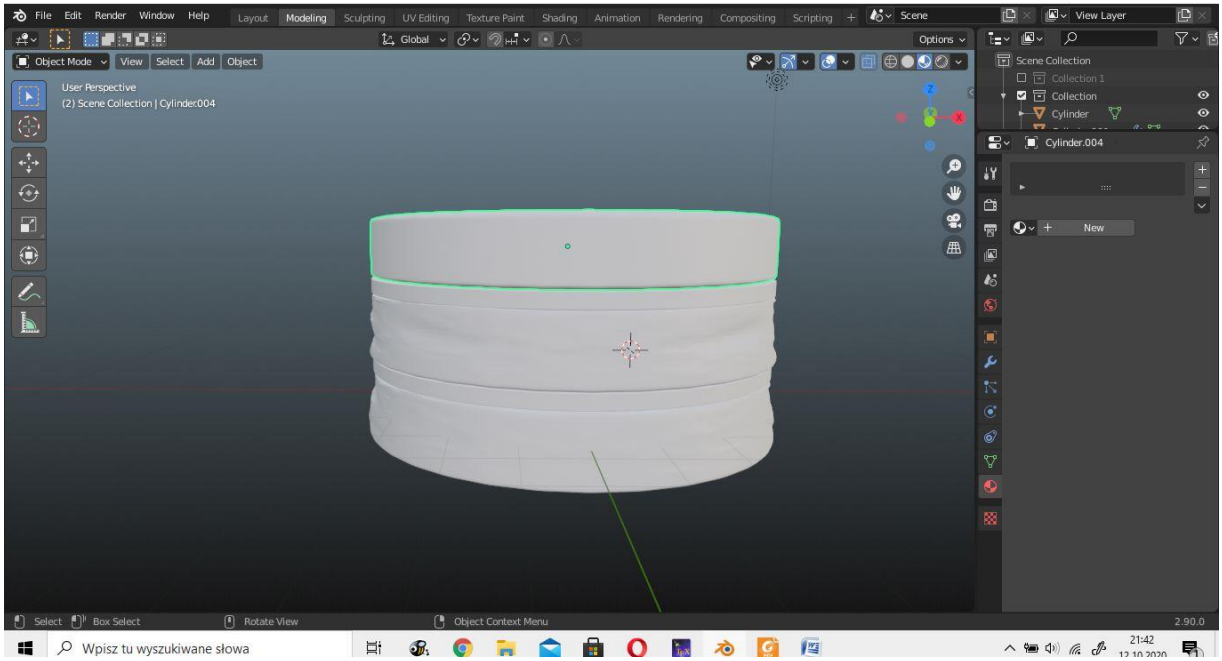
- **Shift + Spacja** Wybór pędzla
- **F** Rozmiar pędzla
- **Shift + F** Siła oddziaływania pędzla
- **Ctrl + F** Kąt pędzla
- **R** Sterowanie kątem
- **E** Wybór trybu pociągnięć
- **B** Maskowanie prostokątne
- **M** Maskowanie pędzlem
- **Alt + M** Wyczyść maskę
- **Ctrl + i** Odwróć maskę
- **H** Ukryj

5. Ćwiczenie 1: Coś na słodko

Ćwiczenie ma na celu utrwalenie zestawu najbardziej użytecznych funkcji Blendera w wersji 2.9, w celu modelowania, nadawania cech materiałowych i renderowania obiektów o cechach organicznych. Przykład przedstawiony w instrukcji dotyczy tortu. Przedstawione ilustracje mają charakter poglądowy. Główne cele ćwiczenia, poza nabyciem umiejętności posługiwania się narzędziami dostarczanymi przez oprogramowanie, to indywidualne podejście (autorskie) do tematu oraz możliwie realistyczne odwzorowanie obiektu przy renderingu.

5.1 Modelowanie podstawowej geometrii obiektu

1. Uruchom oprogramowanie Blender 2.9.
2. Zapisz projekt pod nazwą ImięNazwisko.blend.
3. (Shift + A) Zbuduj scenę złożoną z „plastrów" z których zbudowany będzie tort. Trzy warstwy biszkoptu, połączone masą, wprowadzane jako „Cylinder" 5.1 .



Rysunek 5.1. Warstwy tortu.
Źródło: opracowanie własne.

4. (F2) Każdy obiekt indywidualnie nazywamy. Strukturę sceny możemy również pogrupować w kolekcje w celu łatwiejszego zarządzania projektem (PPM na „Scene Collection" – „New Collecion"). Obiekty przeciągamy do odpowiednich kolekcji 5.2.
5. Wprowadź zagęszczenie siatki „Subdivision surface" 5.3, 5.4.
6. (Tab) W trybie edycji dodaj dwie krawędzie, aby zachować kształt walca z lekko zaokrąglonymi krawędziami 5.5.

Rysunek 5.2. Okno kolekcji ”Outline”.
Źródło: opracowanie własne.

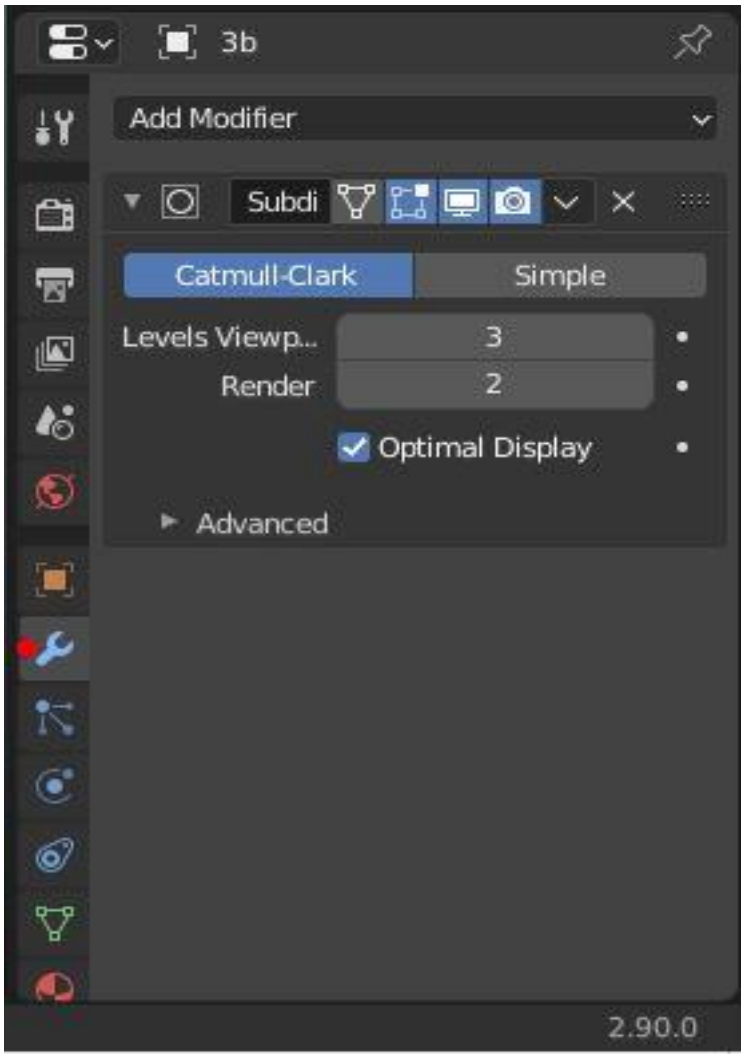
7. Opcje zagęszczenia dobieramy metodą prób i błędów, aż uzyskamy pożądany rezultat. Nie zatwierdzamy jeszcze narzędzia ”Subdivision surface”.
8. (Shift +D) Kopiujemy wierzchnią warstwę tortu.
9. (P) Oddzielamy skopiowany fragment, jako nowy obiekt – polewa 5.6.
10. Nadajemy grubość polewie. Modyfikatory – narzędzie ”Solidyfy” 5.7.
11. W trybie edycji modyfikujemy warstwę polewy, tak aby „oblewała górną część tortu. Pomocne będą narzędzia ”Inset” oraz opcje przyciągania punktów do powierzchni przy edycji proporcjonalnej ”Proportional Editing” 5.8

5.2 Modyfikacje zaawansowane – rzeźbienie

Po uzyskaniu wyniku modelowania zbliżonego do czekolady czy polewy rozlanej w naturalny sposób na powierzchni tortu, pora na dodanie realizmu temu obiektowi. Kluczem jest przemodelowanie szczegółów z wykorzystaniem narzędzi rzeźbiarskich (tryb ”Sculpt mode” 5.9). Dobrą praktyką, zanim przejdziemy do zabiegów artystycznych, jest skopiowanie obiektów, które mają być edytowane dalej. W przypadku zmiany koncepcji nie będzie można wrócić np. do ustawień związanych z nadaniem grubości czy zagęszczeniem siatki.

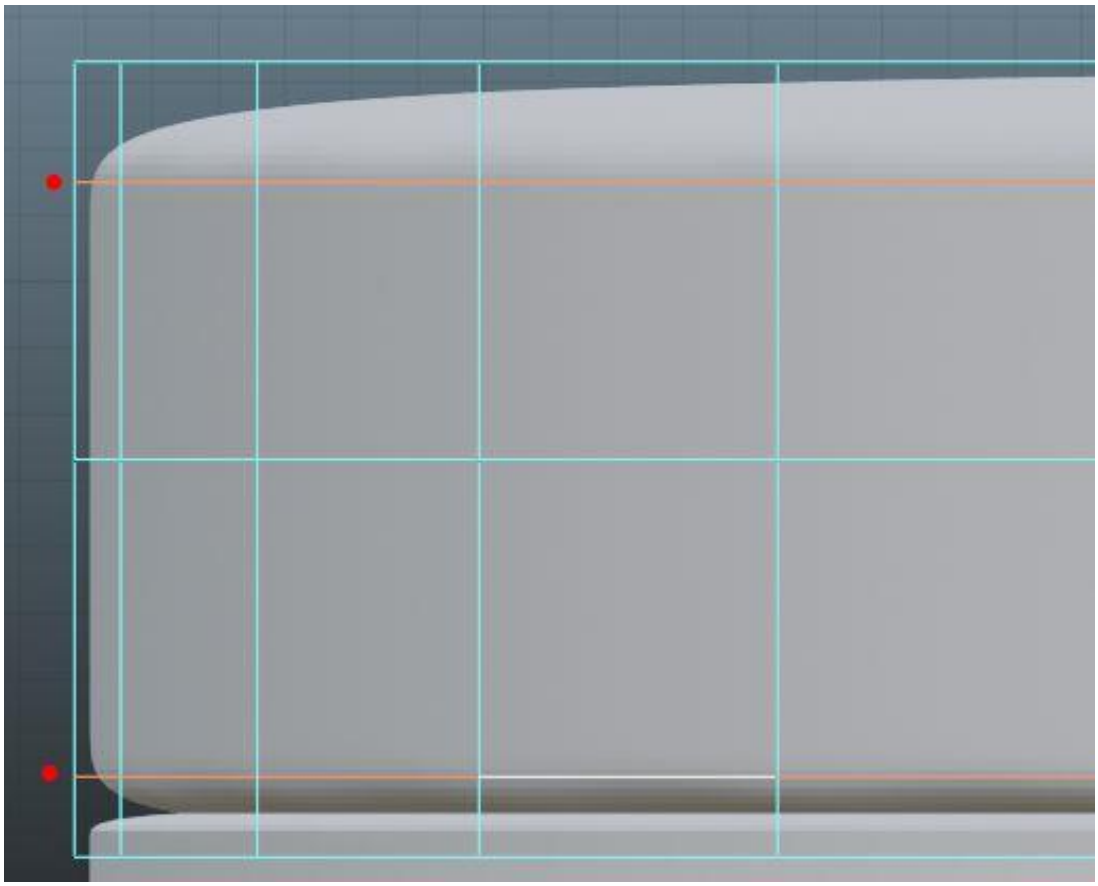
Kopiowanie modelu ”Shift + D” powinno zakończyć się zmianą nazwy i utworzeniem nowej kolekcji – ukrytej w panelu ”Outline” (odznaczony kwadrat po lewej stronie nazwy kolekcji).

1. Po skopiowaniu oryginałów, zatwierdzamy wszystkie modyfikatory 5.10. Powinniśmy mieć do dyspozycji dobrze zagęszczoną siatkę, jak na rysunku 5.11.
2. W trybie rzeźbienia należy zadbać o takie szczegóły, jak krople na krawędziach spłyniętej czekolady 5.12, nieregularności na krawędziach itp.
3. Rozbuduj scenerię i sam tort w większym stopniu niż na rysunku 5.13.

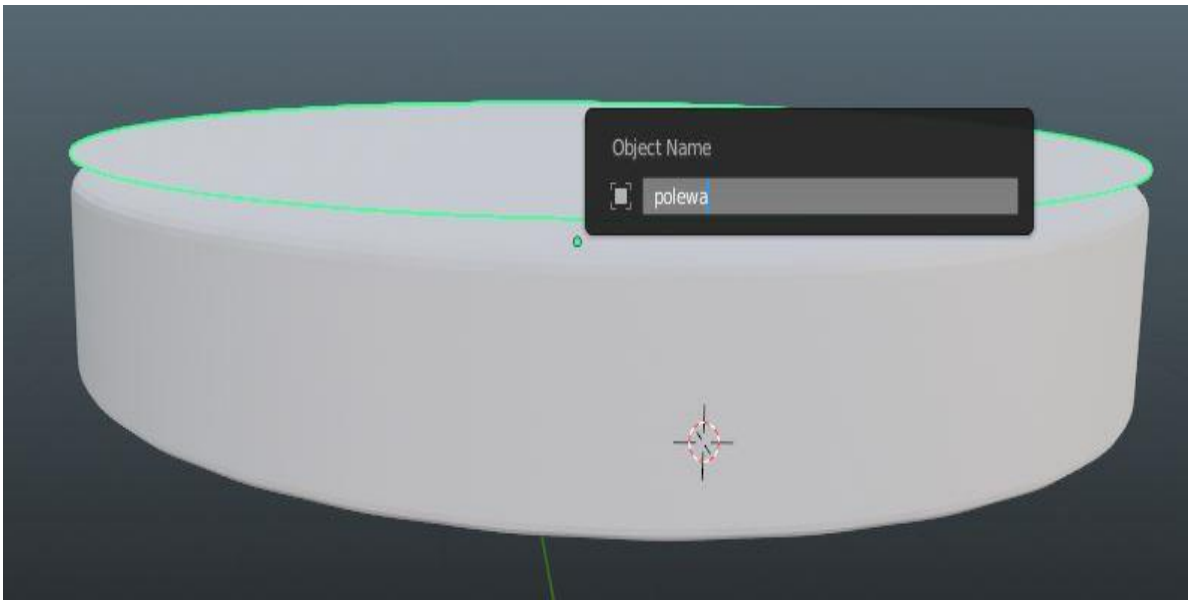


Rysunek 5.3. Zagęszczenie siatki – modyfikator.
Źródło: opracowanie własne.

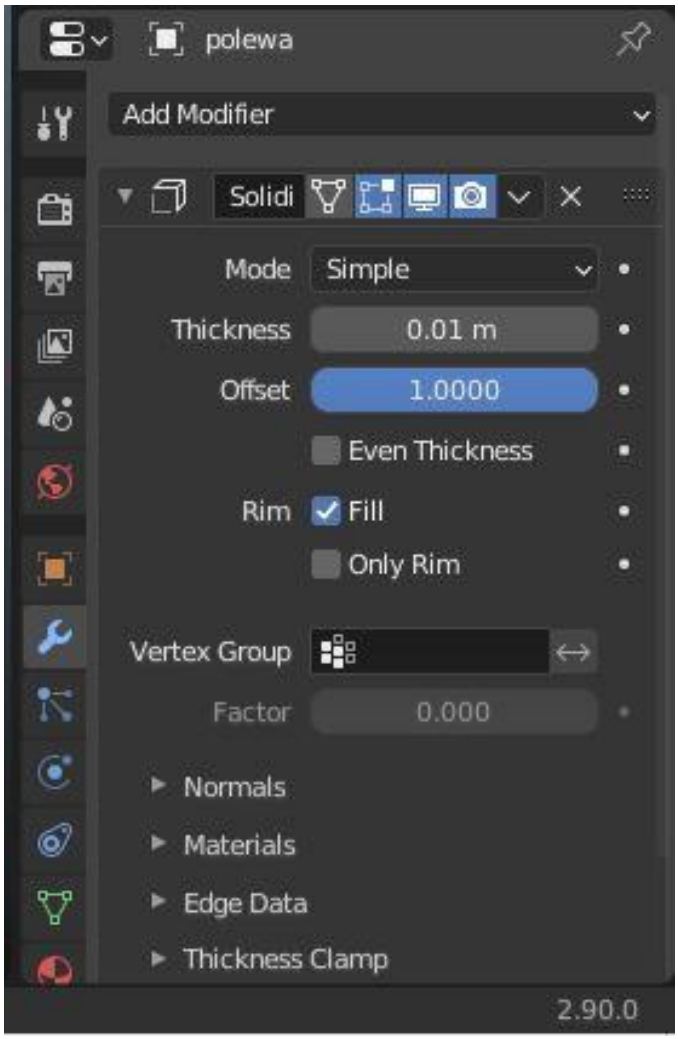
Rysunek 5.4. Modyfikatory.
Źródło: opracowanie własne.



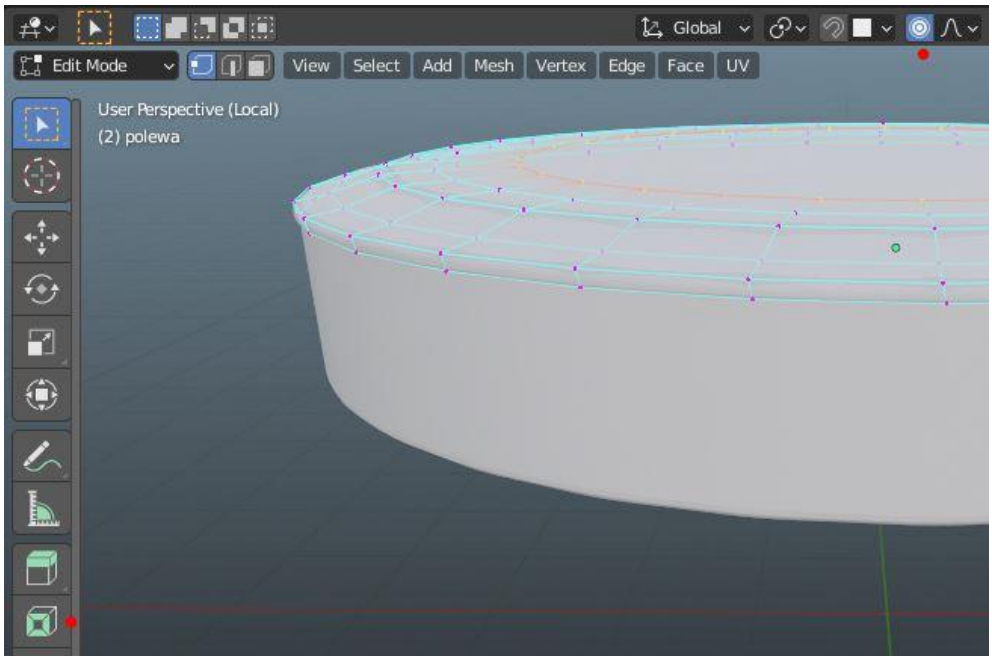
Rysunek 5.5. Krawędzie dodatkowe.
Źródło: opracowanie własne.



Rysunek 5.6. Polewa.
Źródło: opracowanie własne.

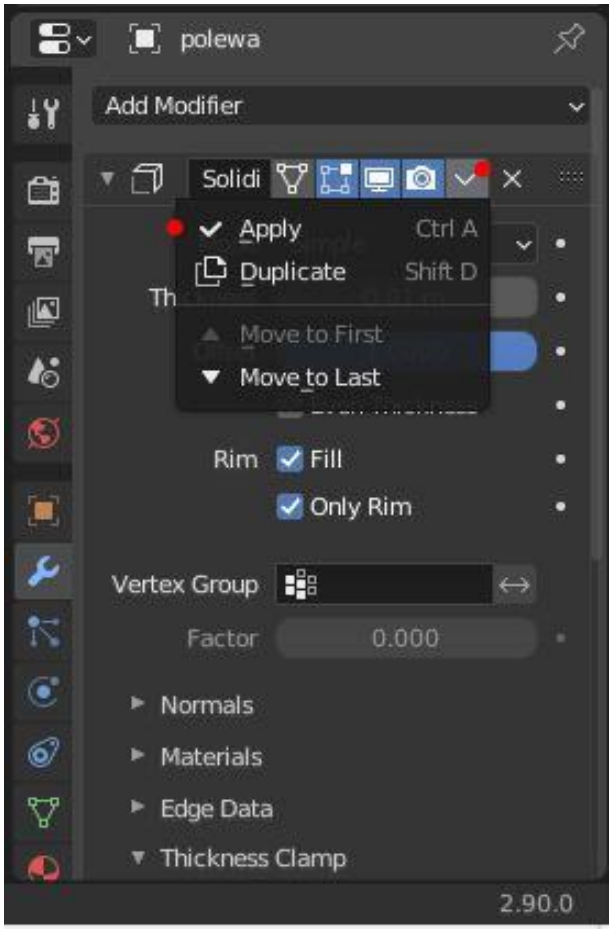


Rysunek 5.7. Nadawanie grubości.
Źródło: opracowanie własne.

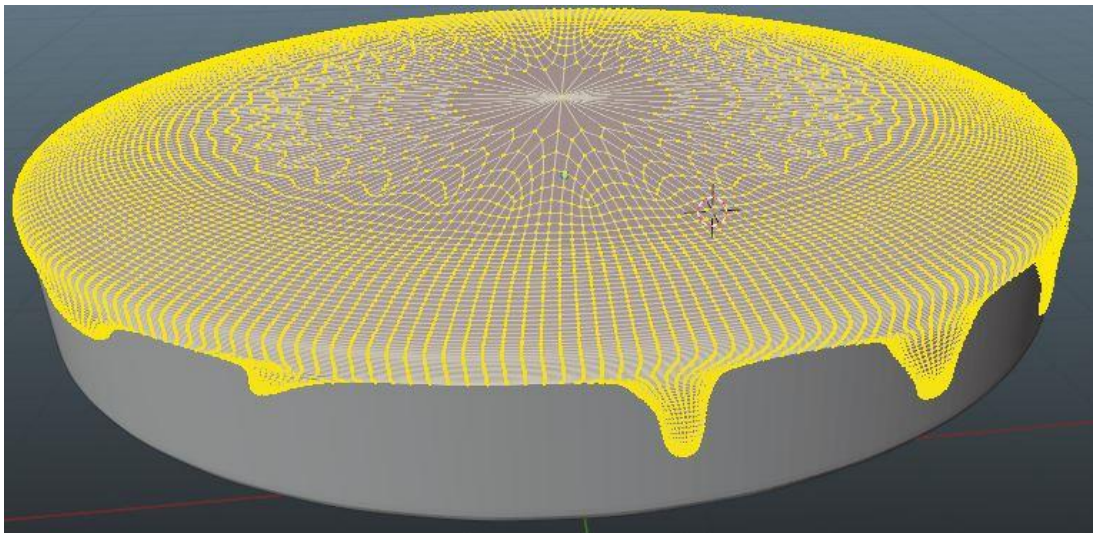


Rysunek 5.8. Edycja polewy.
Źródło: opracowanie własne.

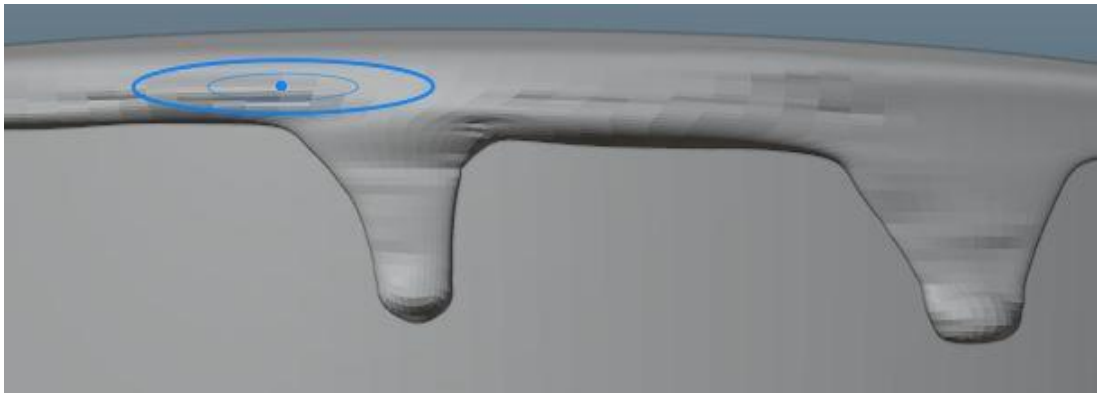
Rysunek 5.9. Włączenie trybu rzeźbienia.
Źródło: opracowanie własne.



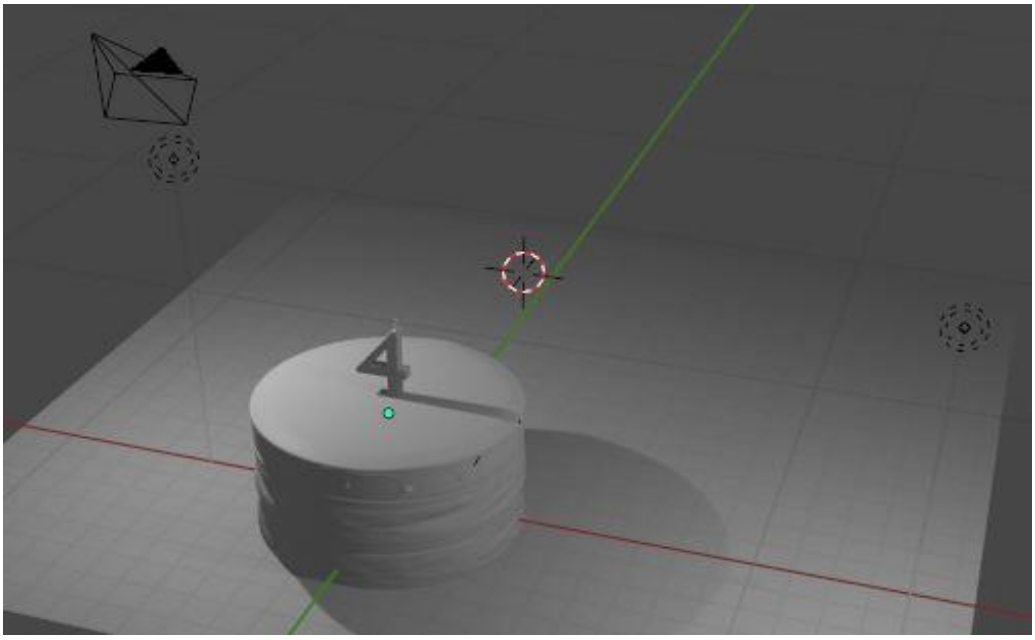
Rysunek 5.10. Zatwierdzenie modyfikatorów.
Źródło: opracowanie własne.



Rysunek 5.11. Siatka po operacjach rzeźbiarskich.
Źródło: opracowanie własne.



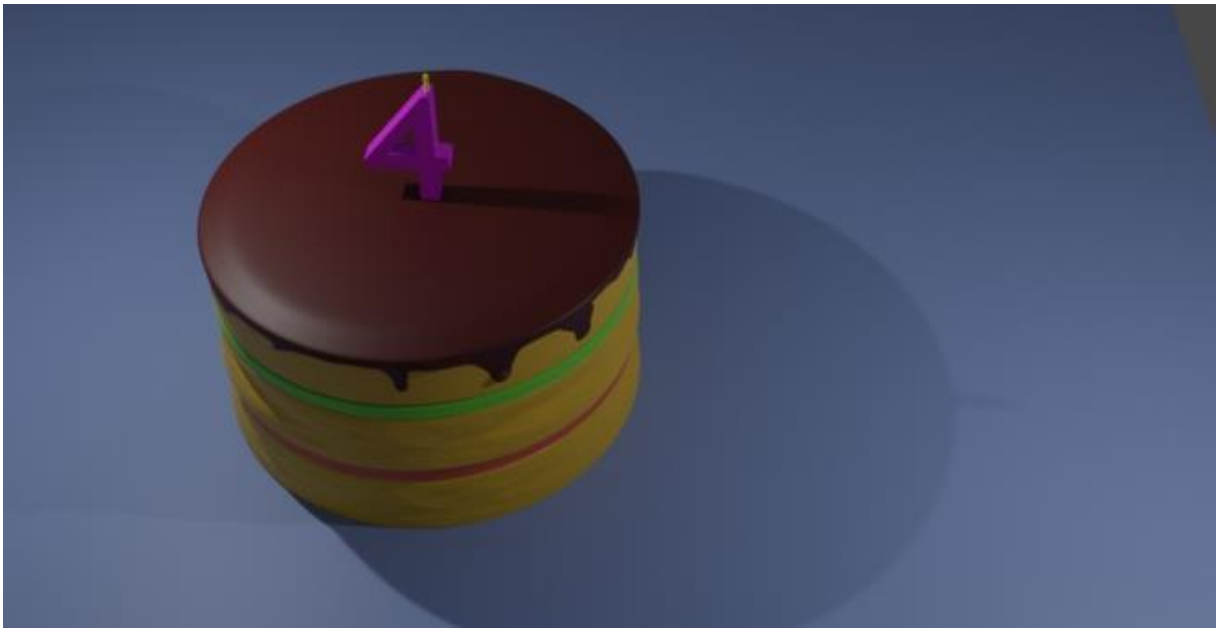
Rysunek 5.12. Krople polewy.
Źródło: opracowanie własne.



Rysunek 5.13. Wygląd warstw.
Źródło: opracowanie własne.

5.3 Podstawowy render

1. Dodaj do obiektów materiały.
2. Zmodyfikuj oświetlenie.
3. Ustaw kamerę.
4. Wykonaj rendering (F12) 5.14.
5. Zapisz projekt.



Rysunek 5.14. Render – tort.
Źródło: opracowanie własne.

6. Ćwiczenie 2: Tekstura – oświetlenie – render

Jako kontynuację ”Ćwiczenia 1” w tym rozdziale zaprezentowano podstawowe funkcjonalności oprogramowania w zakresie nadawania cech materiałowych obiektom, ustawieniami oświetlenia i reakcji obiektów na zróżnicowaną ekspozycję. Efektem finalnym powinno być uzyskanie satysfakcjonującego obrazu po procesie renderingu.

6.1 Silniki renderujące

W pierwszym kroku należy zastanowić się nad silnikiem renderującym, jaki zamierzamy użyć. Część ustawień związanych z materiałami jest współdzielona pomiędzy różnymi silnikami renderującymi, są jednak takie, które np. w jednym działają, a w innym nie są obsługiwane. Aby zaoszczędzić sobie problemów, dobrze jest zdecydować się na początku na któryś z wariantów i konsekwentnie się go trzymać.

Podstawowe trzy silniki renderujące, wykorzystywane w Blenderze, to: Blender internal, Cycles oraz Eevee. Poniżej przedstawiono różnice między tymi silnikami renderującymi, ich zalety i wady. Przegląd stanowi skrótowe podsumowanie efektów uzyskiwanych przy ich stosowaniu.

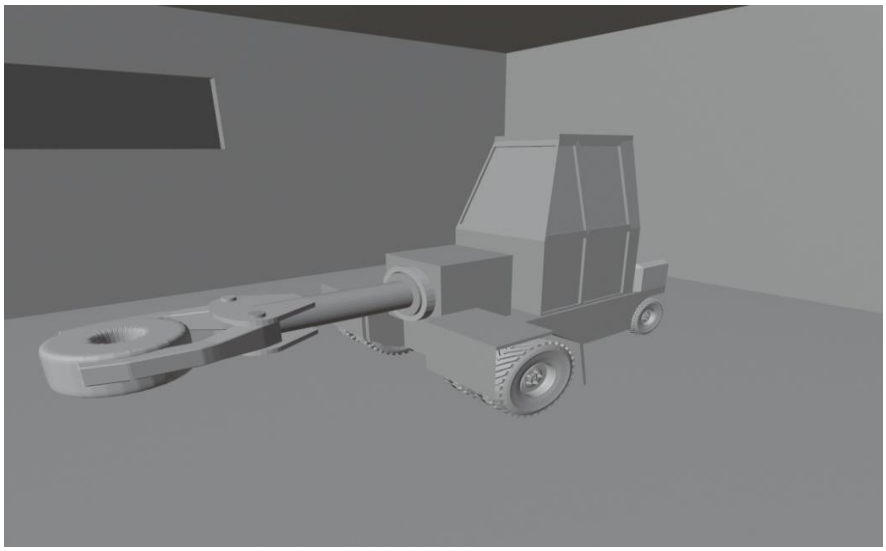
1. Blender Render (Blender internal):
   - jest to oryginalny silnik renderujący Blendera z kodem źródłowym z początku lat 90.;
   - obsługuje zaawansowane funkcje, takie jak śledzenie promieni (ray traycing), rozpraszanie światła pod powierzchnią materiału, odbicia światła, uproszczona funkcja globalnego oświetlenia;
   - jest to bardzo szybki silnik renderujący, przy czym większość jego funkcji charakteryzuje się renderowaniem nierealistycznym, dobrej jakości efekty wizualne są trudne do uzyskania i wymagają sporego nakładu pracy. Fotorealizm jest możliwy do osiągnięcia jedynie przez zastosowanie tricków;
   - szybkie renderowanie możliwe dzięki Blender internal niestety wiąże się z prostym oświetleniem i podstawowymi ustawieniami materiałów i tekstur;
   - Blender internal jest świetną opcją dla każdego rodzaju nierealistycznego renderowania. Może to oznaczać m.in. grafikę ruchową, animacje kreskówkowe i wizualizacje informacji.
2. Cycles:
   - jest to ogromny krok naprzód pod względem realistycznego renderowania, z pełnoprawnym globalnym oświetleniem i fizycznie dokładnymi obliczeniami (wprowadzony w 2011 roku);
   - znalazł on swoje miejsce w takich komercyjnych oprogramowaniach, jak: Cinema 4D i Rhino;
   - bez wątpienia cycles to najlepszy wybór do realistycznego renderowania. Ponieważ jest oparty na zasadach fizycznych, jest bardzo łatwy w użyciu z artystycznego punktu widzenia;
   - cycles oferuje przyspieszenie GPU, które może drastycznie skrócić czas renderowania. Korzystanie z zasobów karty graficznej może okazać się lepszym wyborem niż pozostawianie obliczeń procesorowi komputera;
   - obsługuje Ray traycing w stopniu pozwalającym w szybki sposób osiągnąć zadowalające rezultaty przy renderingu obiektów wykonanych np. ze szkła.

3. Eevee
    - Eevee to nowy fizyczny mechanizm renderujący w czasie rzeczywistym. Działa zarówno jako mechanizm renderujący dla końcowych klatek, jak również silnik napędzający okienko widokowe Blendera;
    - posiada zaawansowane funkcje, takie jak wolumetryczność, odbicia i załamania, rozpraszanie podpowierzchniowe, miękkie i kontaktowe cienie, głębia ostrości, rozmycie ruchu;
    - z uwagi na wysoką wydajność przy renderowaniu w czasie rzeczywistym Eevee znajduje zastosowanie przy opracowaniu gier komputerowych;
    - nie obsługuje funkcji Ray traycing – konsekwencją są trudności z odwzorowaniem materiałów wykonach np. ze szkła. Brak obsługi śledzenia promieni wchodzących w interakcje ze szkłem utrudnia realistyczne odwzorowanie efektu soczewkowego.

Prosta sceneria bez nałożonych tekstur ukazuje różnice w efekcie renderingu z wykorzystaniem przytoczonych silników. W przypadku użycia silnika Blender internal – w ustawieniach w wersji 2.9 nosi on nazwę "Workbench" – zauważyć można płaską scenerię 6.1, niskiej jakości odwzorowanie oświetlenia i znikome oddziaływanie cieni generowanych przez oświetlenie.

Dla rezultatów osiągniętych przy wykorzystaniu silnika Eevee, wzrasta głębia scenerii, zauważyć można silniejsze oddziaływanie oświetlenia na obiekty znajdujące się w otoczeniu manipulatora kuźniczego przedstawionego na rysunku 6.2. Dla promieni światła padających z zewnątrz pomieszczenia przez okno (na podłodze pod wysięgnikiem manipulatora) zauważamy nierealistyczne – ostre odcięcie między oświetloną a nieoświetloną powierzchnią. Brakuje oddziaływania oświetlenia rozproszonego.

Efekt uzyskany przy pomocy Cycles przedstawiono na rysunku 6.3, oświetlenie oraz cieniowanie są na wysokim poziomie, bez dodatkowych ustawień osiągamy rezultaty przewyższające BI i Eevee. Oświetlenie podłogi przez światło wpadające przez okno charakteryzuje się miękkimi krawędziami. Duży kontrast pomiędzy zacienionymi obszarami oświetlonymi dodaje scenerii głębi.



Rysunek 6.1. Render BI.
Źródło: opracowanie własne.

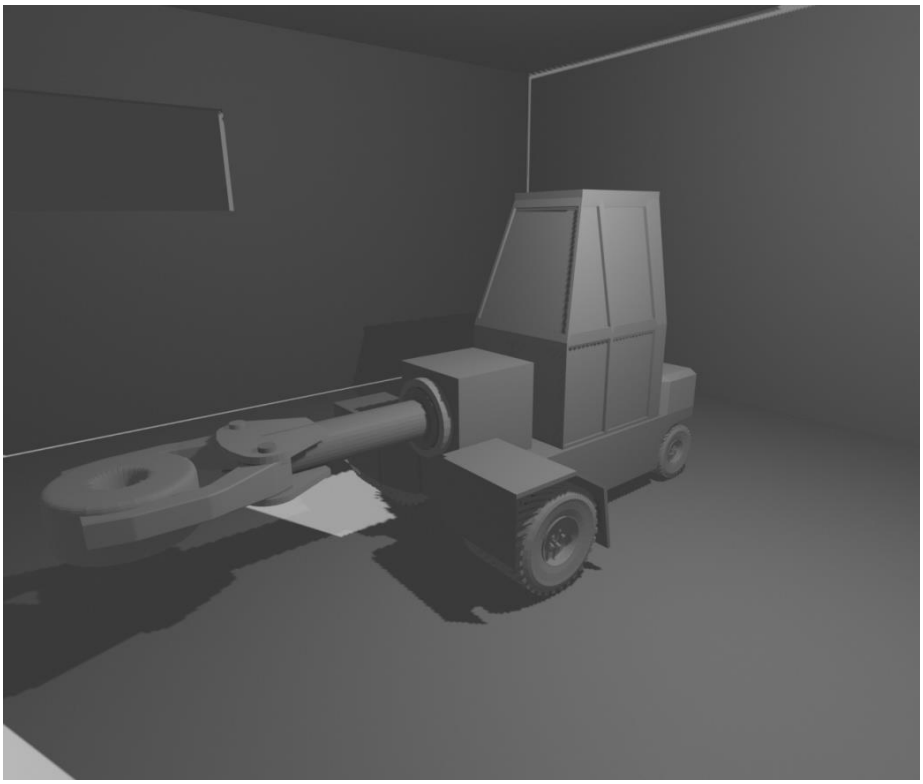
Nadmienić należy dodatkowo, że w przypadku Eevee i Cycles konieczne było czterokrotne zwiększenie mocy lamp użytych w scenerii.

6.2 Ustawienia oświetlenia od otoczenia

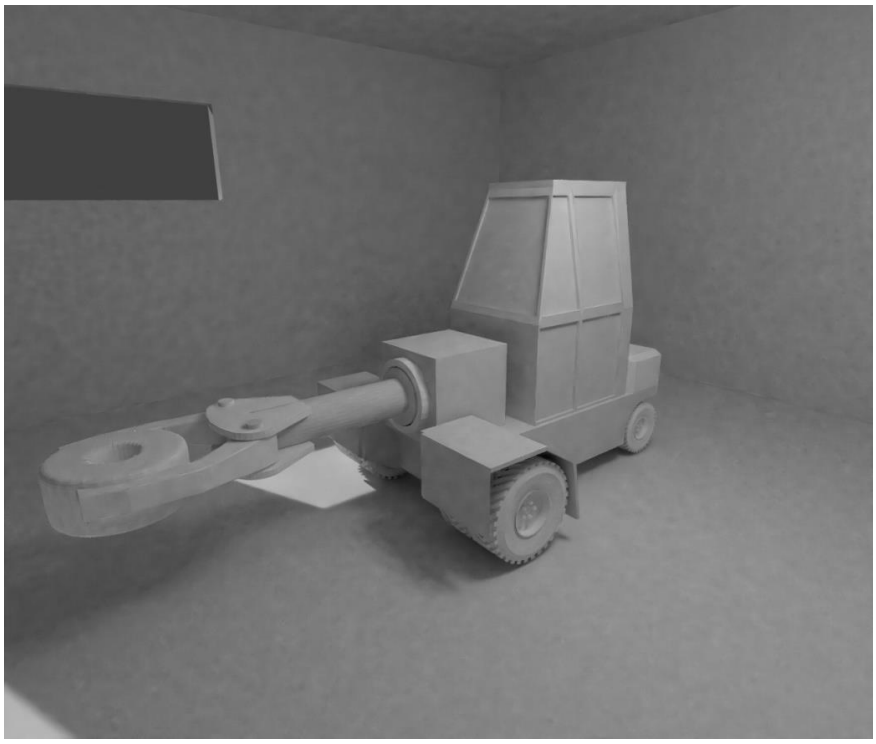
Realne oświetlenie charakteryzuje się tym, że promienie padające od źródła światła na różnego typu powierzchnie rozpraszają się i odbijają od tych powierzchni, doświetlając zacienione obszary (uzyskujemy wtedy miękkie cienie). Efekt ten uzyskujemy, włączając funkcję ”Ambient Occlusion” 6.4. W ustawieniach, zależnie od scenerii, dobieramy wartość ”Distans”, określającą, jak daleko od siebie mają znajdować się obiekty, aby oświetlony obiekt ”doświetlał” obszary zacienione drugiego obiektu. ”Faktor” określa stopień mieszania się promieni odbitych i tych bezpośrednio oświetlających obiekt. ”Trace precision” to dokładność odwzorowania. Dodatkowo opcje ”Bent normals” i ”Bounces Approximation” powinny być włączone w celu realistycznego przeliczania trajektorii fotonów odbijających się od powierzchni oświetlonych.

Kolejnym przydatnym ustawieniem jest ”Screen Space Reflection”, określający załamania i odbicia światła na obiektach. Z reguły domyślne ustawienia w tym przypadku wystarczają. Domyślnie chropowatość materiału, przy której odbicia nie są wyliczane (Max Roughness) ustawione jest na 0,5, czyli w połowie pomiędzy lustrzanym wykończaniem powierzchni a doskonale absorbującym światło.

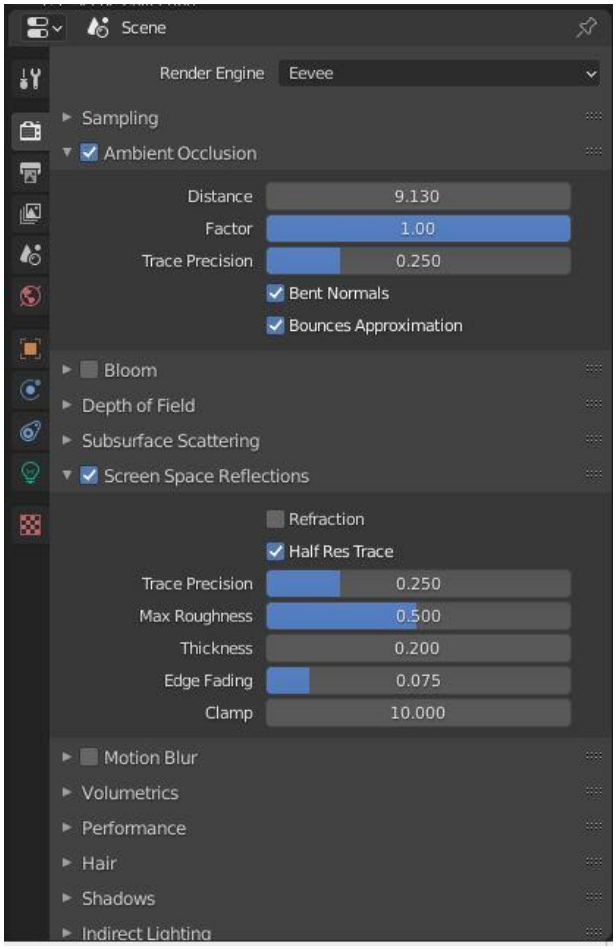
W celu precyzyjnego wyliczenia oddziaływania rozproszonego oświetlenia na obiekty w scenerii, konieczne jest dodanie czujników oświetlenia. Wybieramy skrót klawiszowy ”Shift + A”, obiekt, który musimy wstawić, to ”Irradiance Volume”. Należy dopasować wymiary obszaru oczujnikowanego dla scenerii (zachowując margines zapasu), jak na rysunku 6.5. W zakładce ”Indirect Lighting” wybieramy opcję ”Bake Indirect Lighting” – rysunek 6.6. Cieniowanie i oświetlenie zostaną wyliczone dla naszej scenerii. Pozostaje tylko uzupełnić scenerię o źródła światła – skrót klawiszowy ”Shift + A” opcja ”Light” i z menu wybieramy jedno ze źródeł światła. Podstawowe ustawienia to ”Power” – moc oświetlenia oraz ”Color”.



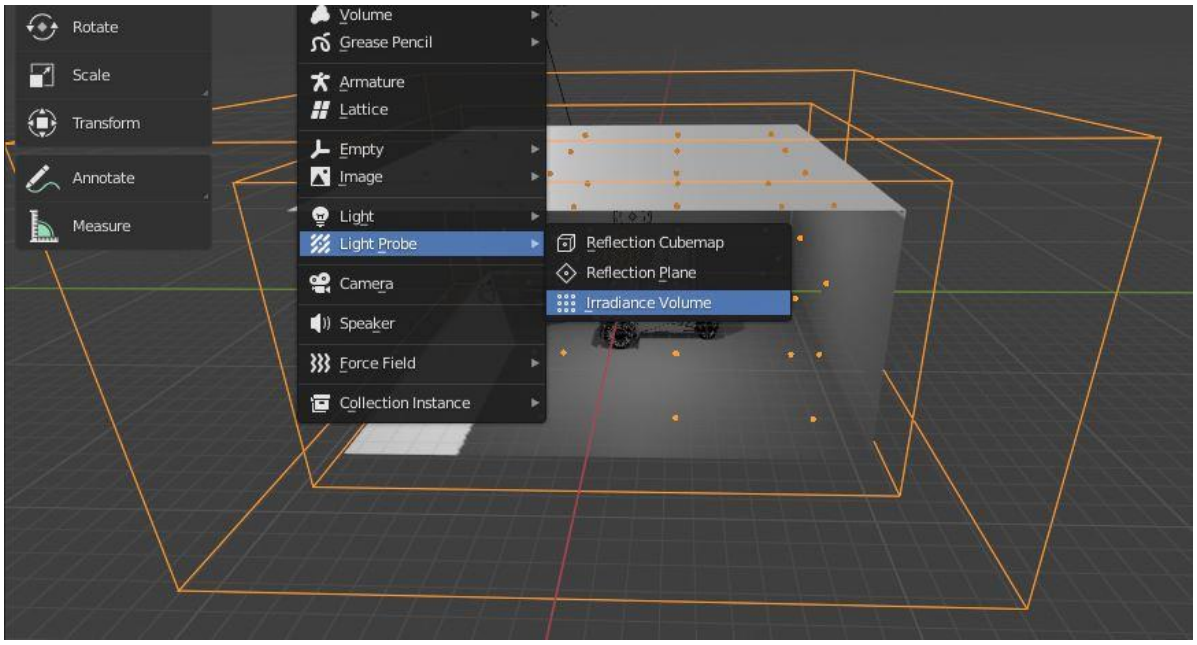
Rysunek 6.2. Render Eevee.
Źródło: opracowanie własne.



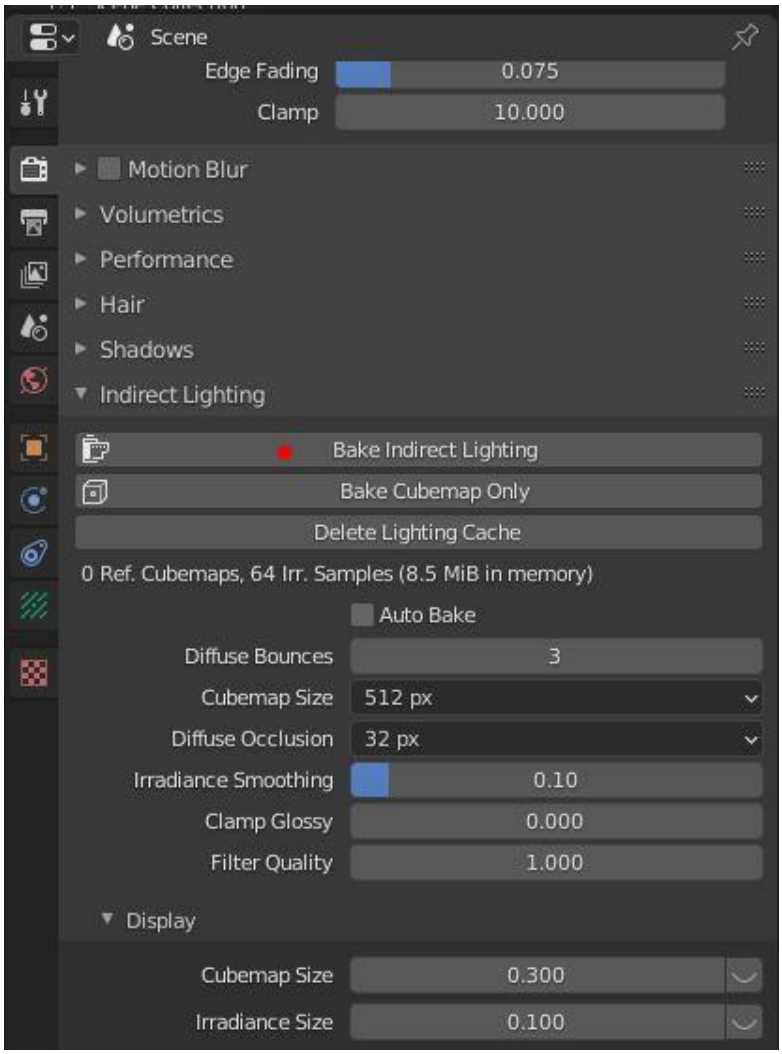
Rysunek 6.3. Render Cycles.
Źródło: opracowanie własne.



Rysunek 6.4. Ambient Occlusion.
Źródło: opracowanie własne.



Rysunek 6.5. Irradiance Volume.
Źródło: opracowanie własne.



Rysunek 6.6. Indirect Lighting.
Źródło: opracowanie własne.

6.3 Parametry materiałowe

Nadawanie cech materiałowych do obiektów to jedna z ważniejszych czynności z punktu widzenia uzyskania realistycznych efektów w scenerii. Każdy materiał w otaczającym nas świecie posiada szereg cech, które w mniejszym lub większym stopniu należy odwzorować podczas modelowania, a są to:

- Barwa – kolorystyka obiektów polega nie tylko na jednolitym pokryciu powierzchni kolorem, często na powierzchni występują zmiany w odcieniu, wzory, desenie itp.
- Błędy kształtu – obiekty o idealnych (przewidywalnych) kształtach co prawda również występują w rzeczywistości, jednak anomalia i losowe zmiany geometrii dodają naturalności scenerii. Efekt losowych zmian możemy uzyskiwać na wiele sposobów, m.in.: modelowanie (dodawanie punktów i ich transformacja), metody rzeźbiarskie (Sculpting mode), tekstury proceduralne (wykorzystanie danych w plikach graficznych do wygenerowania modyfikacji w strukturze powierzchni).
- Oddziaływanie z otoczeniem – sposób, w jaki obiekty reagują na padające na nie światło (ilość światła odbitego, penetracja fotonów w głąb materiału itp.).
- Emisyjność – część obiektów może generować światło, np. rozgrzany metal.

Efektywną metodą wprowadzania cech materiałowych obiektom jest operowanie tzw. ”Node Editor”. Istnieje wiele stron internetowych, dostarczających pakiety tekstur przygotowanych do użycia w programach graficznych, takich jak Blender. Część z nich udostępnia tekstury w wysokiej rozdzielczości za darmo. Przykładem tego może być strona https://cc0textures.com.

Przygotowanie do nałożenia tekstury wymaga zaopatrzenie się w zestaw plików graficznych, przygotowanych w ten sposób, aby oprogramowanie było w stanie z każdego z nich wydobyć odpowiednie cechy powierzchni. Bazując np. na zdjęciu powierzchni, jesteśmy w stanie, korzystając z oprogramowania do edycji grafiki (np. Gimp), zastosować filtry np. w celu uwidocznienia cieni na powierzchni sfotografowanego obiektu, które są efektem jej nierówności. Taki obraz konwertowany na skalę szarości posłużyć może do wygenerowania nierówności na modelowanym obiekcie.

Gotowy zestaw grafik zdejmuje z nas konieczność często żmudnego procesu edycji zdjęć, ustawień obrazów w ”Node editor” oraz korekt w grafice źródłowej, jeśli efekt nie jest zadowalający.

W zestawie tekstur na ogół można spotkać się z następującymi typami grafik:

- Color – mapa koloru, stanowiąca podstawowe wybarwienie obiektu.
- Roughness – chropowatość powierzchni. Praktyczne zastosowanie to wskazanie, które obszary mają wysoką chropowatość (pochłaniają światło), a które mają cechy refleksyjne spowodowane niską chropowatością (odbijają fotony). Obraz przygotowany jest w skali szarości, im bardziej kolor zbliża się do białego, tym bardziej spada chropowatość, a obiekt bardziej odbija światło. Dla odcieni zbliżających się do czarnego – chropowatość wzrasta.
- Displacement – określa mapę przemieszczeń w skali makroskopowej (pozwala na zróżnicowanie wysokości obszarów na powierzchni).
- Normal – mapa definiujące ustawienie wektora normalnego do powierzchni w każdym punkcie.
- Ambient Occlusion – ta składowa tekstura stanowi uzupełnienie mapy kolorów o ”pogłębienie” cieniowania. Obszary ciemne będą przyciemnione (bardziej zacienione).

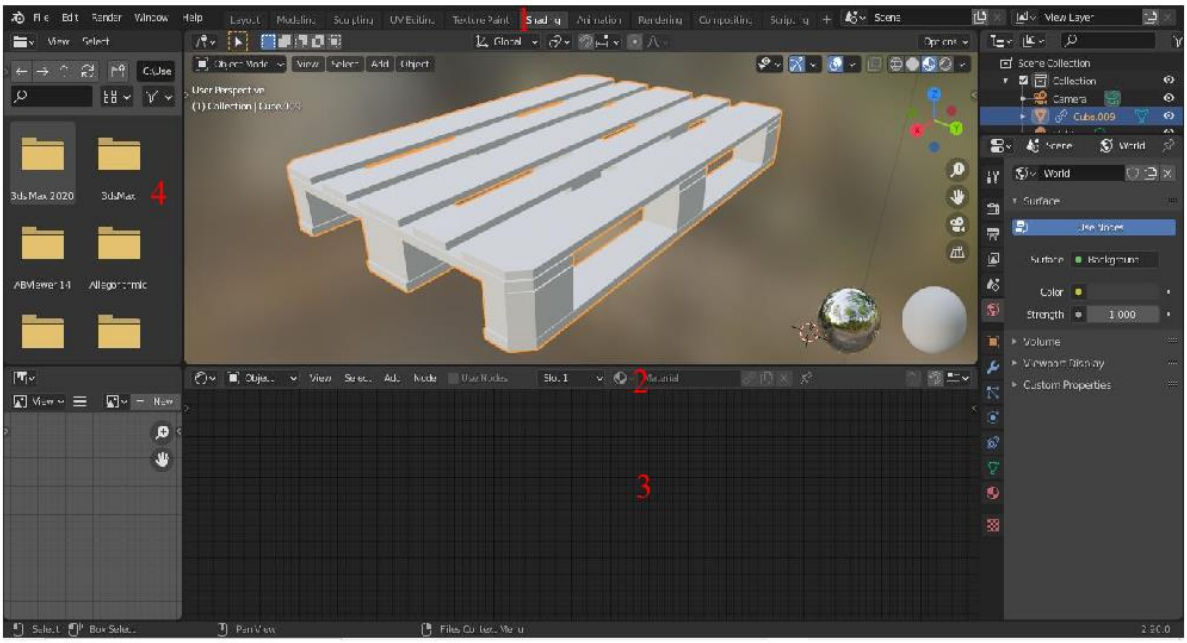
- Metalness – grafika definiująca, które osary są metaliczne (kolor biały), a które nie (kolor czarny).
- Opacity/Mask – maskowanie lub mapa czarno-biała określająca krycie.
- Emission – emisyjność powierzchni, czyli definicja obszarów emitujących światło.

Skorzystanie z ”paczki” tekstur na ogół ogranicza się do dodania standardowego układu węzłów w obszarze ”Node editor”. Przełączenie się w tryb ”shader” 6.7

Przytoczony przykład stanowi przedstawienie podstawowych metod nadawania cech materiałowych w ”Node editor”, z wykorzystaniem gotowych zestawów tekstur PBR (Physically Based Rendering), czyli metodzie pozwalającej na realistyczne odwzorowanie interakcji oświetlenia z powierzchnią.

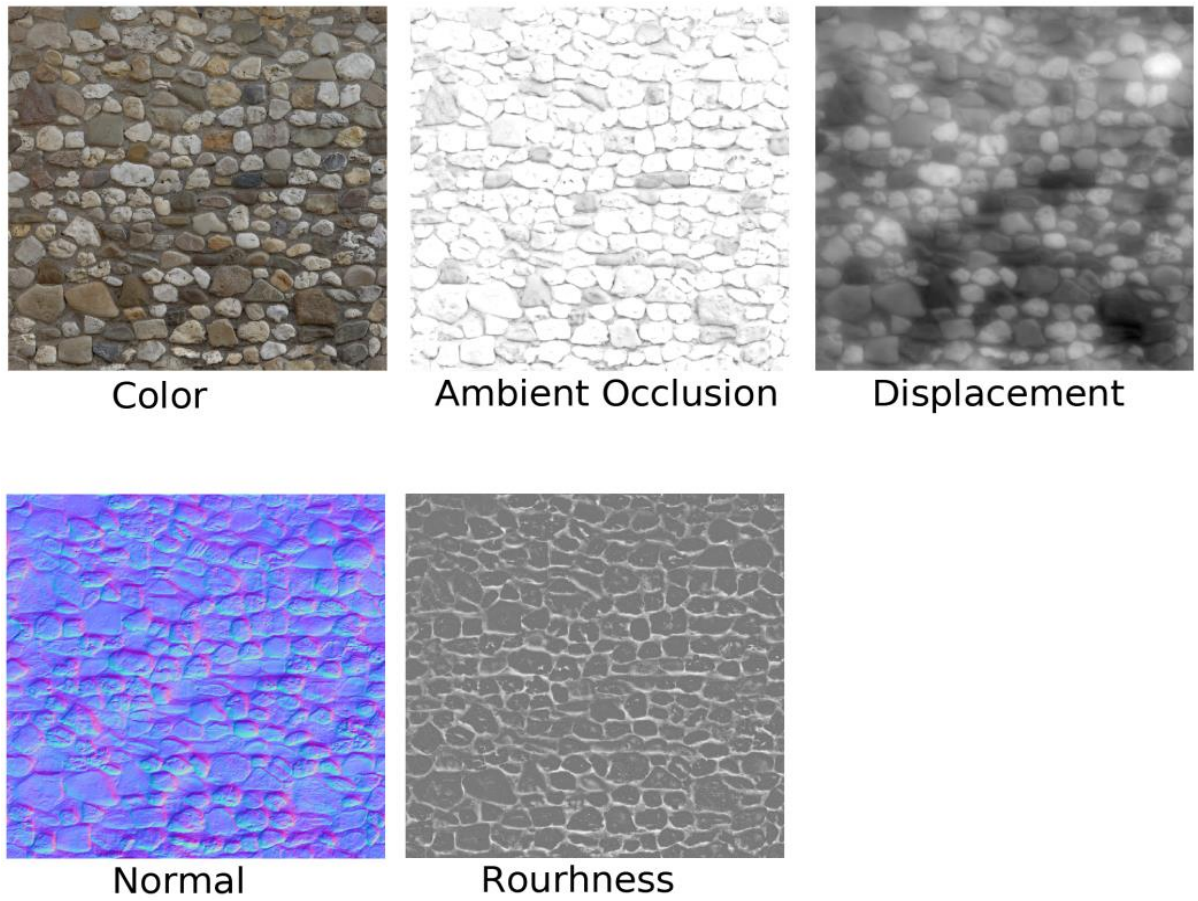
6.4 Rozpoczęcie racy z teksturami i podstawowe ustawienia

W pierwszej kolejności włączamy tryb pracy ”Shader” 6.7, aby uzyskać podgląd wyglądu materiału oraz zestaw okien do edycji ustawień materiałowych. Obiektem, który został użyty w tym przykładzie jest drewniana europaleta. Materiał proceduralny, jaki wykorzystano, to ”Planks021”, który pobrany został ze strony https://cc0textures.com.



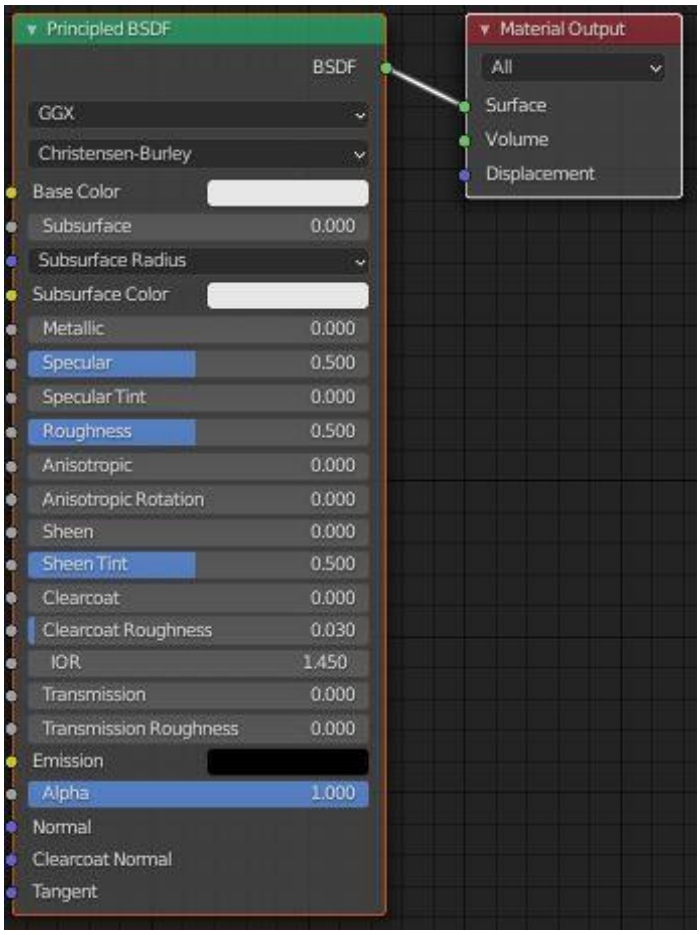
Rysunek 6.7. Tryb shader.
1 – wybór trybu pracy ”shader”; 2 – dodawanie nowego materiału do obiektu; 3 – okno edycji węzłów „node editor”; 4 – eksplorator plików.
Źródło: opracowanie własne.

Po rozpakowaniu archiwum uzyskujemy dostęp do zestawu grafik przedstawionych na rysunku 6.8

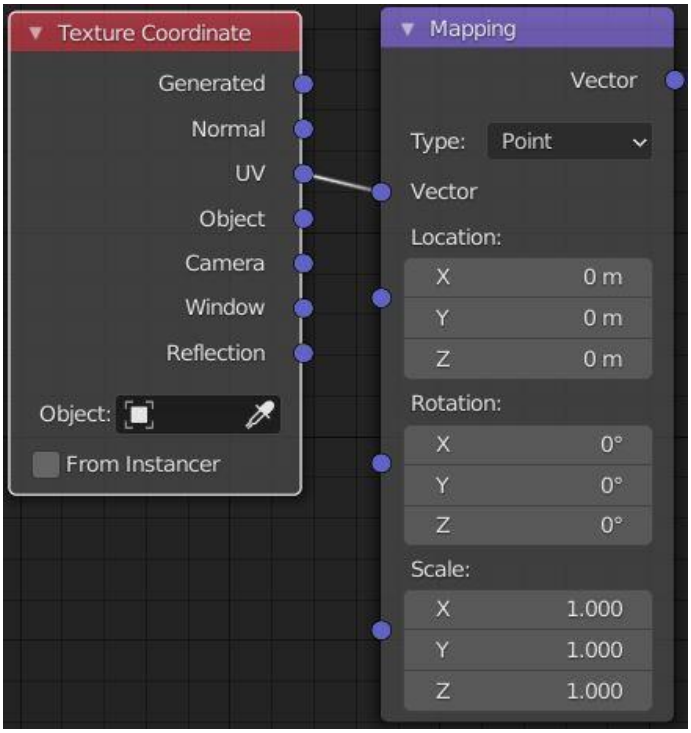


Rysunek 6.8. Zestaw tekstur PBR.
Źródło: opracowanie własne.

1. Po wyborze trybu pracy zaznaczamy obiekt do nałożenia tekstury, włączamy nowy materiał – rysunek 6.7. W oknie ”Node editor” pojawią się dwa węzły: Principed BSDF – podstawowe ustawienia koloru, bazujące na modelu Disney, a także Material Output – przesyła informacje z edytora węzłów do obiektu w scenerii.
2. Zbudowanie modułu do zarządzania położeniem tekstury ułatwia ustalenie pozycji obrazów względem obiektu w scenerii. Wprowadzamy w tym celu węzły: Texture, Coordinate i Mapping. Łączymy UV (rozwinięcie siatki obiektu) z Vector (koordynaty siatki). Moduły dostępne są w zakładce ”Add” – ”Input” oraz ”Vector”. W pracy z obiektami w ”Node editor” ważne jest zwrócenie uwagi na kolorystykę węzłów łączących obiekty (muszą zgadzać się typy informacji przekazywanych między obiektami).

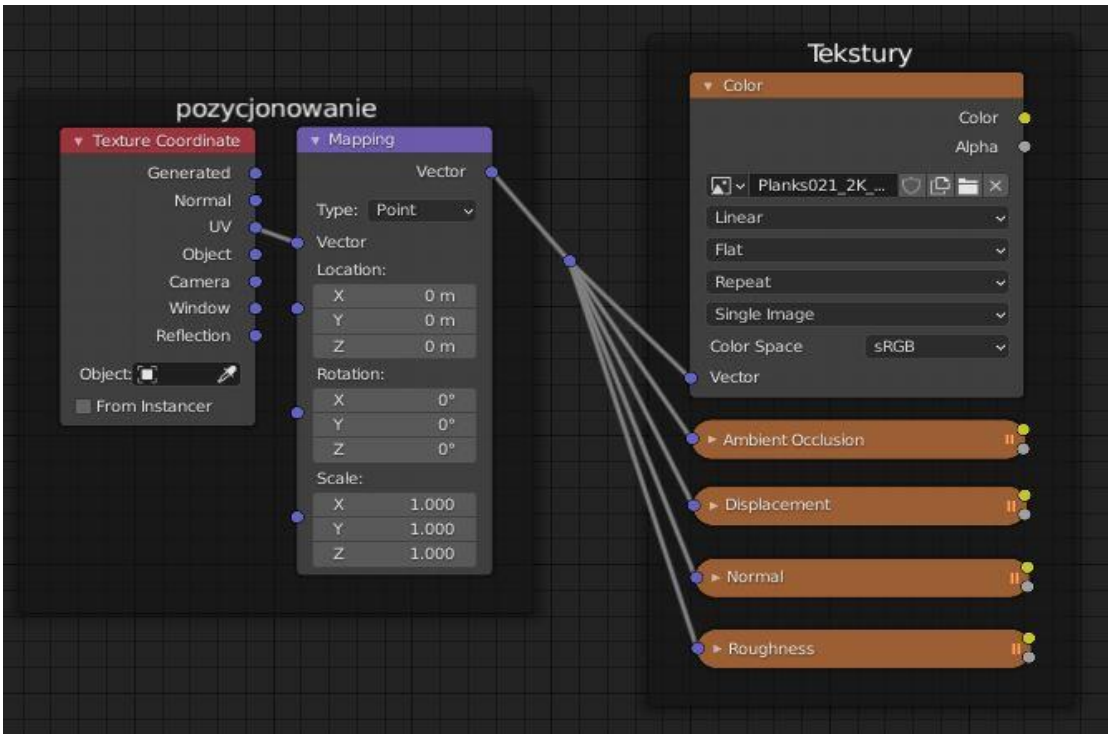


Rysunek 6.9. BSDF.
Źródło: opracowanie własne.



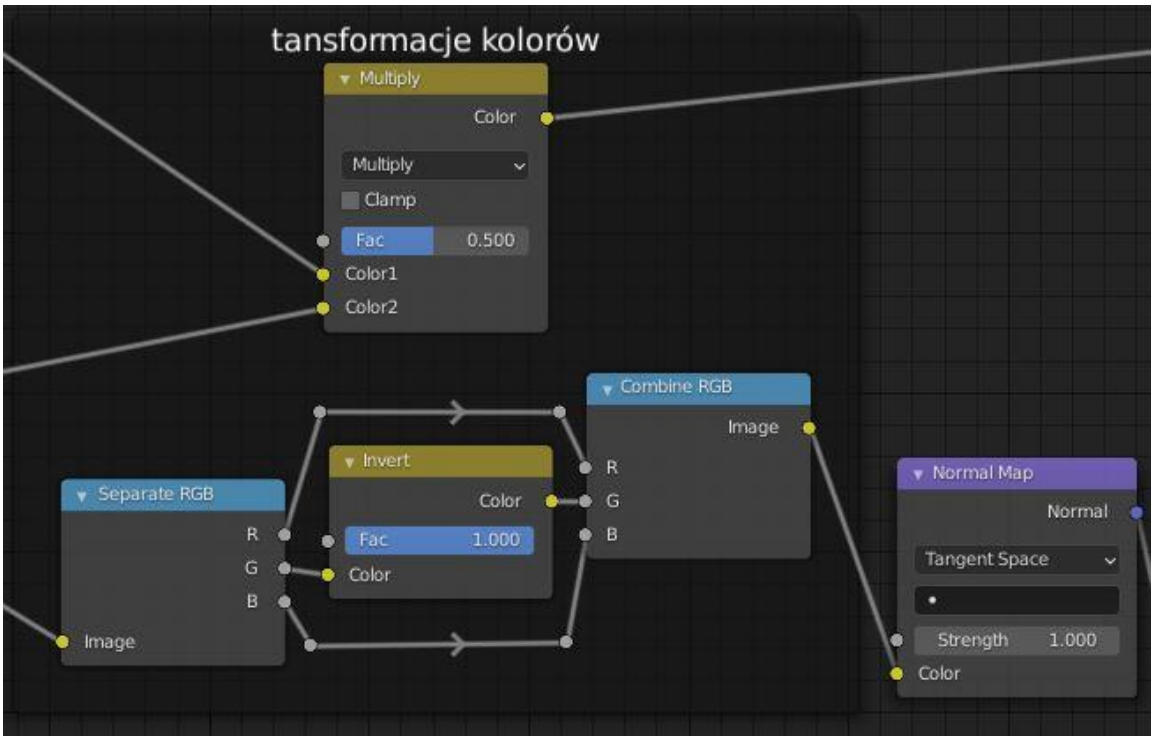
Rysunek 6.10. Położenie tekstury.
Źródło: opracowanie własne.

3. W kolejnym kroku umieszczamy w oknie grafiki źródłowe. Najprostsza metoda to przeciągnięcie obrazu z okna eksploratora 6.7 na obszar ”Node editor”. Alternatywnie można dodać obrazy, klikając ”Add” – ”Texture” – ”Image Texture” i na podanym panelu dodajemy grafikę, podając jej lokalizację na dysku. Do każdego z okien wprowadzamy informację o położeniu – ”Vector”. W ustawieniach ”Input Texture” zmieniamy opcję ”Color Space” na ”Non-Color” dla map odpowiadających na wektory normalne, metaliczność powierzchni oraz chropowatość.
4. Obiekty w ”Node editor” można indywidualnie nazywać (F2) i grupować tematycznie (Ctrl + G). Po zmianie ustawień w danym węźle można opcje zminimalizować symbolem trójkąta w lewym górnym narożniku okienka węzła. Aranżację danych wejściowych do generowania przedstawiono na rysunku 6.11.



Rysunek 6.11. Wprowadzenie tekstur do Node Editor.
Źródło: opracowanie własne.

5. Pozostaje powiązanie danych wejściowych z ”Principed BSDF”. Dwa źródła danych wymagają szczególnej uwagi. W przypadku kolorystyki obiektu konieczne jest połączenie dwóch źródeł: Color oraz Ambient Occlusion za pomocą ”MixRGB” do następnego z ”Add” – ”Color” (opcja Multyply). Tak uzyskane dane podpinamy pod ”Base Color” w ”Principed BSDF”.
   Drugi strumień danych wymagający modyfikacji to mapa wektorów normalnych. Tekstury dostępne u większości dostawców opracowane są w standardzie DirectX, przy czym Blender funkcjonuje w standardzie OpenGL. Konsekwencją jest błędne kalkulowanie składowej Y wektora normalnego do powierzchni. Aby naprawić pojawiające się błędy, istnieją dwie drogi: inwersję zielonej składowej obrazu w zewnętrznym programie graficznym lub zastosowanie analogicznej transformacji kolorów z wykorzystaniem narzędzi Blendera. W tym celu należy:
   - rozdzielić strumień koloru na składowe ”Add” – ”Converter” – ”Separate RGB”;

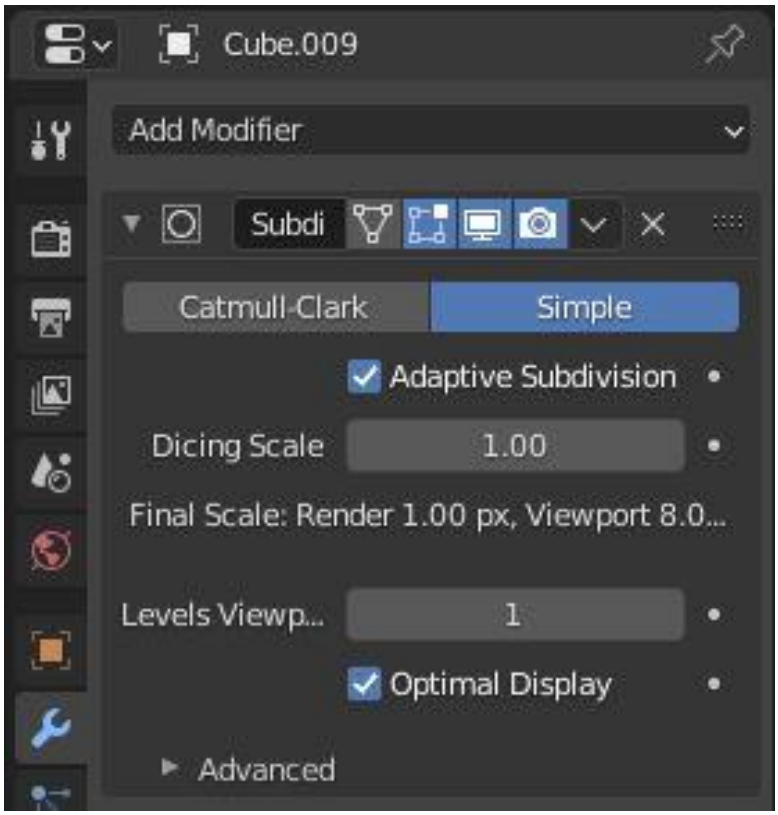


Rysunek 6.12. Transformacja strumieni koloru.
Źródło: opracowanie własne.

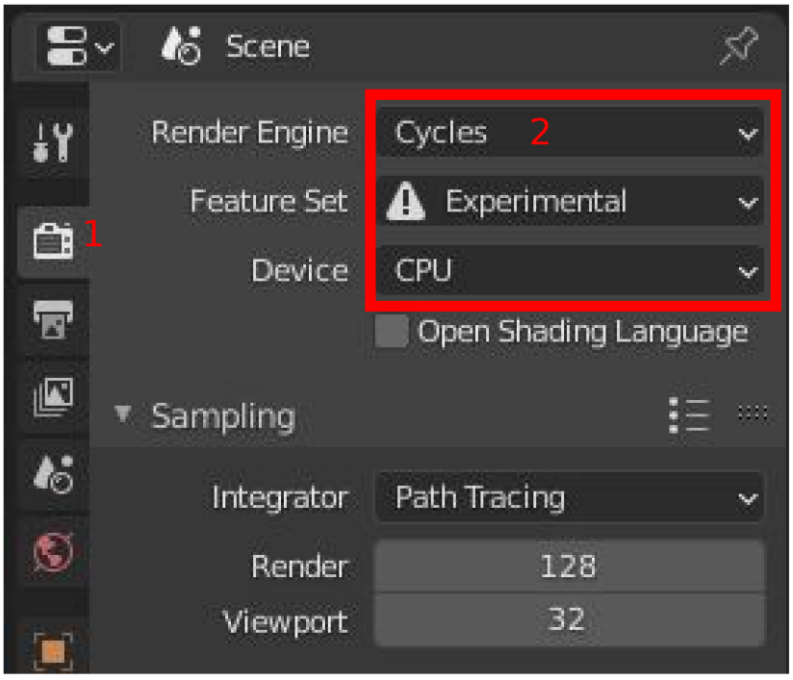
- odwrócić zielony kanał: ”Add” – ”Color” – ”Invert”;
- połączyć trzy składowe: ”Add” – ”Converter” – ”Combine RGB”.

Tak uzyskane dane można podłączyć bezpośrednio pod pin ”Normal” w ”Principed BSDF” lub (jeżeli chcemy mieć dodatkową kontrolę nad intensywnością uzyskanego efektu) za pośrednictwem ”Add” – ”Vector” – ”Normal Map”. Układ węzłów ukazano na rysunku 6.12

6. W celu wygenerowania przemieszczeń poszczególnych punktów obiektu, tak, aby ich realne położenie pokrywało się z obrazem tekstury (pomijając możliwość ”ręcznego” przemodelowania powierzchni, korzystając z narzędzi edycyjnych oraz modułu rzeźbiarskiego – ”Sculpting”), łączymy składową tektury ”Displacemnt” przez moduł ”Add” – ”Vector” – ”Displacement”, bezpośrednio z odpowiednim pinem w module ”Material output”. Dodatkowo, dla lepszych rezultatów, można zmienić silnik renderujący 6.14 pkt. 1 na ”Cycles” 6.14 pkt. 2, opcje ”Feature Set” – ”Experimental” oraz o ile dysponujemy mocną kartą graficzną rendering za pomocą ”GPU”.
   Problemem może być dogęszczenie siatki (zbyt małe nie odda nierówności powierzchni, zbyt duże zaabsorbuje zasoby sprzętowe przy renderingu), adaptacyjne dogęszczenie w modyfikatorach 6.13 pozwala zautomatyzować proces. Dodatkowo w opcjach materiałowych należy włączyć widoczność przemieszczeń – opcja ”Displacement” 6.15. Efekt zastosowanych modyfikatorów i modułów z ”Node Editor” przedstawiono na rysunku 6.16. Widoczne są tu oprócz tekstury, interakcje pomiędzy obiektem a oświetleniem: zacienienia oraz refleksy uzależnione od odkształceń obiektu.
7. Co prawda, tekstura została już nałożona na obiekt, jednak domyślne ułożenie grafiki na obiekcie powoduje losowe przypisanie tekstury do poszczególnych powierzchni. W celu precyzyjnego dopasowania tekstur do obiektu stosuje się tryb ”UV Editing” 6.17.

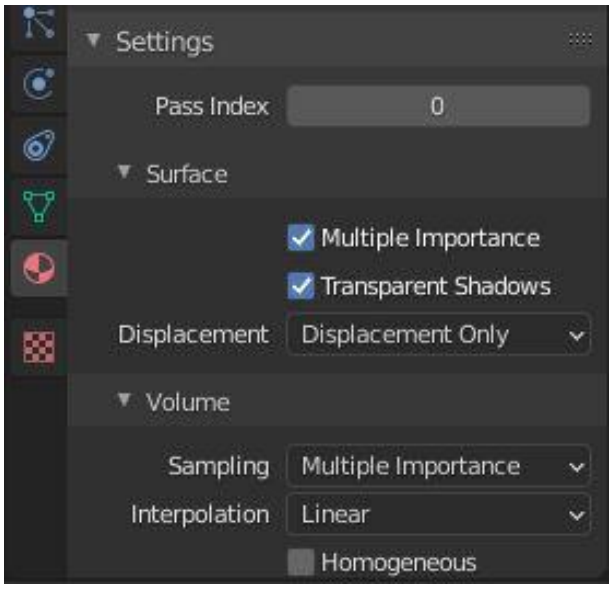


Rysunek 6.13. Modyfikator Subdivision Surface.
Źródło: opracowanie własne.



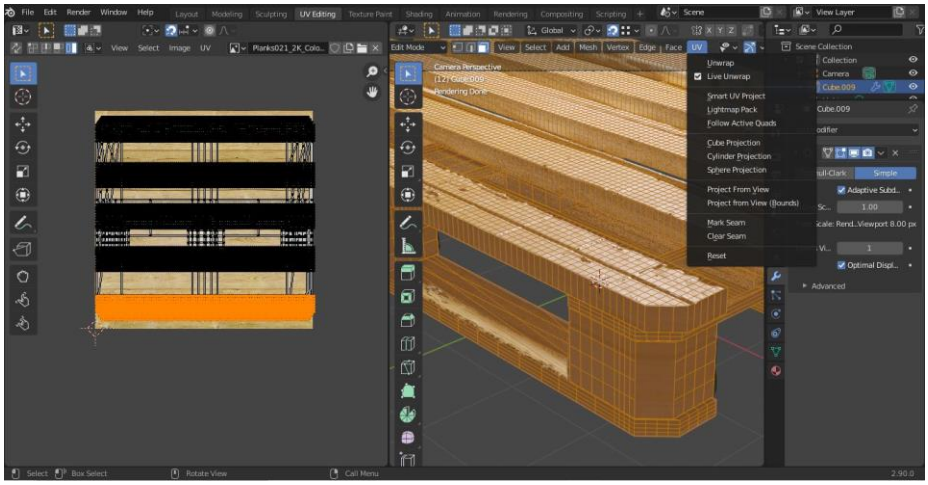
Rysunek 6.14. Ustawienia silnika renderującego.
Źródło: opracowanie własne.

W rozważanym przypadku użyto narzędzia ”UV” – ”Cube projection”, które rozwinęło siatkę obiektu (widoczną w lewym oknie) i nałożyło ją na obraz z teksturą. Każdą z powierzchni można indywidualnie zaznaczyć, przemieszczać oraz skalować tak, aby elementy geometryczne pokrywały się z właściwymi fragmentami tekstury.



Rysunek 6.15. Opcja materiałowa Displacement.
Źródło: opracowanie własne.

Rysunek 6.16. Render – paleta.
Źródło: opracowanie własne



Rysunek 6.17. UV Editor.
Źródło: opracowanie własne.

7. Wprowadzenie do POV-Ray

Oprogramowanie POV-Ray (Persistence Of Vision Raytracer) jest silnikiem renderującym, który jako podstawę funkcjonowania używa algorytmu śledzenia fotonów. Głównym założeniem twórców oprogramowania było możliwie wierne odwzorowanie i symulowanie realistycznej fizyki oświetlenia stosowanego w modelowanej scenerii.

Dystrybuowane jest ono na licencji AGPL3 i umożliwia darmowe wykorzystanie na wielu płaszczyznach do ram określonych w warunkach licencjonowania.

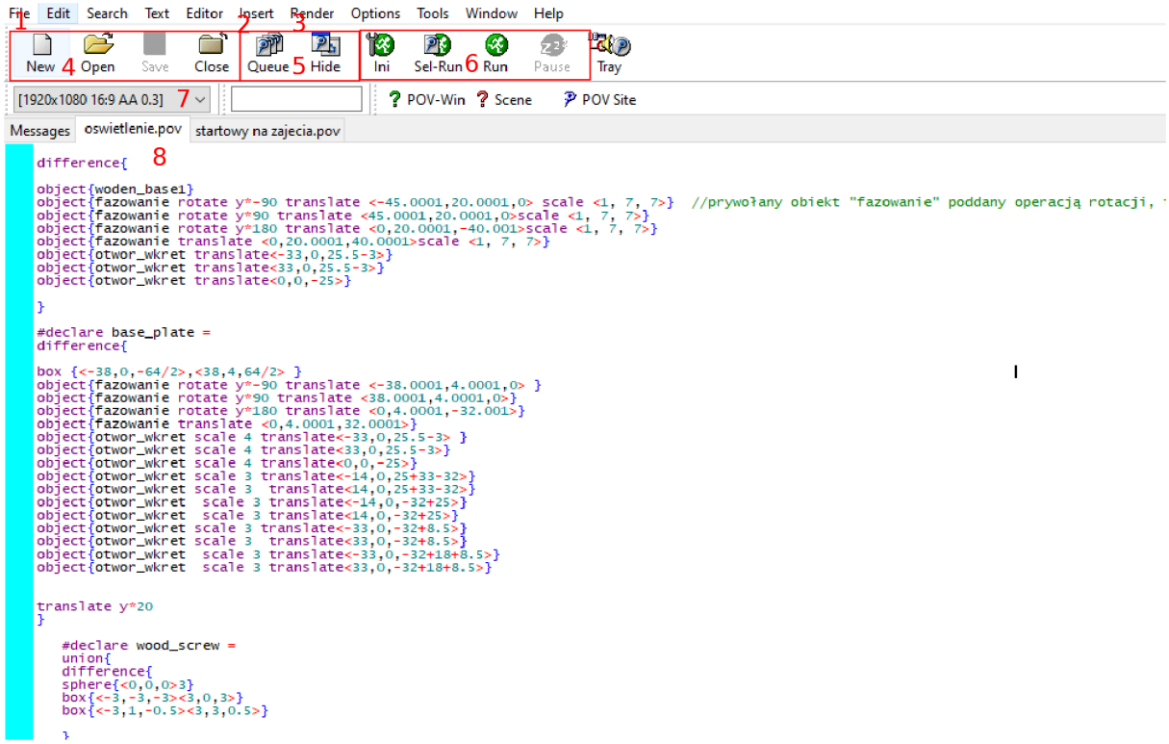
Początki funkcjonowania oprogramowania datuje się na 1992 rok (wersja 1.0). Ostatnia stabilna wersja to 3.7, wydana w 2013 roku. Dokumentację techniczną do oprogramowania opracował POV-Team (2013). Do tej pory, oprócz przytoczonej publikacji, Sifre (2014) przedstawił w swojej książce podstawy do wykorzystania języka POV-Ray w celu wizualizacji obiektów w planimetrii oraz stereometrii w ujęciu czysto matematycznym.

7.1 Interface – POV-Ray

Oprogramowanie posiada rozbudowany interface, o szerokich możliwościach ingerencji w wewnętrzne ustawienia silnika renderującego, dla potrzeb skryptu jednak opis funkcjonalności oprogramowania został zredukowany do najważniejszych zagadnień, istotnych przy modelowaniu, nadawaniu cech materiałowych obiektom i renderowaniu. Zagadnienia opisane w tym rozdziale stanowią praktyczny przewodnik dla celów inżynierskich. Wygląd okna z przygotowaną scenerią przedstawiono na rysunku 7.1.

Podstawowe elementy oprogramowania to:

1. Menu ”File” zawiera opcje związane z otwieraniem, zapisywaniem i drukowaniem plików projektów.
2. Menu ”Insert” – zestaw gotowych schematów programistycznych dotyczących odpowiednio:
    - Ready made scenes – gotowe scenerie, które pozwalają w szybki sposób zorganizować przestrzeń renderowanego obszaru (tło, oświetlenie, kamery, zamglenie itp.);
    - Basic shapes, Shapes – zestaw narzędzi do modelowania przestrzennego. W przypadku wybrania odpowiedniego modułu działanie poszczególnych parametrów ustawianych podczas definiowania obiektów są dodatkowo opisane;
    - CSG operation – zestaw operacji logicznych na obiektach: łączenie obiektów, różnica, część wspólna oraz dodatkowe zaawansowane operacje;



Rysunek 7.1. Okno główne w programie POV-Ray.
Źródło: opracowanie własne.

- Shape modifiers – modyfikacje związane z interakcjami obiektu z innymi elementami scenerii (odbicie, cieniowanie) oraz cechami brył;
- Transformation – modyfikacje obiektu oparte na translacjach, rotacjach oraz skalowaniu;
- Loop – narzędzia szyków liniowych, kołowych oraz wyciągnięć po ścieżkach;
- Math functions – podstawowe funkcje matematyczne;
- Directives – narzędzia służące do budowania składni programu definiującego scenerię, m.in. polecenia deklaracji zmiennych, pętli warunkowych i funkcji logicznych;
- Include files – zestaw dodatków, m.in. do definicji tekstur, materiałów, dodatkowych modyfikatorów obiektów;
- Random – generator losowych modyfikacji, w szczególności dostępne są tutaj generatory trawy i modyfikatory pozycjonowania obiektów w scenerii;
- Animation – definicja animacji ruchowych;
- Color – paleta wykończeń kolorystycznych obiektów definiowanych w skali RGB lub na zasadzie nazewnictwa naturalnego (angielskie nazwy kolorów);
- Texture and Materials – gotowe pakiety podstawowych materiałów, uwzględniających kolorystykę oraz strukturę rzeczywistych materiałów;
- Patterns – podstawowe wzory dwuwymiarowe możliwe do wykorzystania podczas generowania wyglądu obiektów w scenerii;
- Cameras – ustawienia parametrów optycznych kamery;
- Light sources – umieszczanie różnego typu źródeł światła w scenerii;
- Sky, fog, rainbow – umieszczanie dodatkowych efektów w scenerii (m.in.: niebo, mgła, tęcza);

- Radiosity and photons – zaawansowane algorytmy przeliczająca oświetlenie w scenerii;
- Expressions, Statements – dodatkowe moduły matematyczne i szczegółowe ustawienia renderingu.

3. Opcje związanie z przeliczaniem scenerii, kolejkowaniem operacji i wykorzystaniem procesorów.
4. Moduł zarządzania plikami (nowy plik, otwieranie, zapisywanie, zamykanie plików).
5. Kolejka operacji przeliczeniowych podczas renderingu i odkrywanie/ukrywanie efektu przeliczeń scenerii.
6. Ustawienia inicjalizacji obliczeń scenerii oraz uruchomienia/zatrzymania silnika renderującego.
7. Ustawienia rozdzielczości wyjściowej oraz opcji wygładzenia anizotropowego (AA).
8. Obszar programu definiującego scenerię.

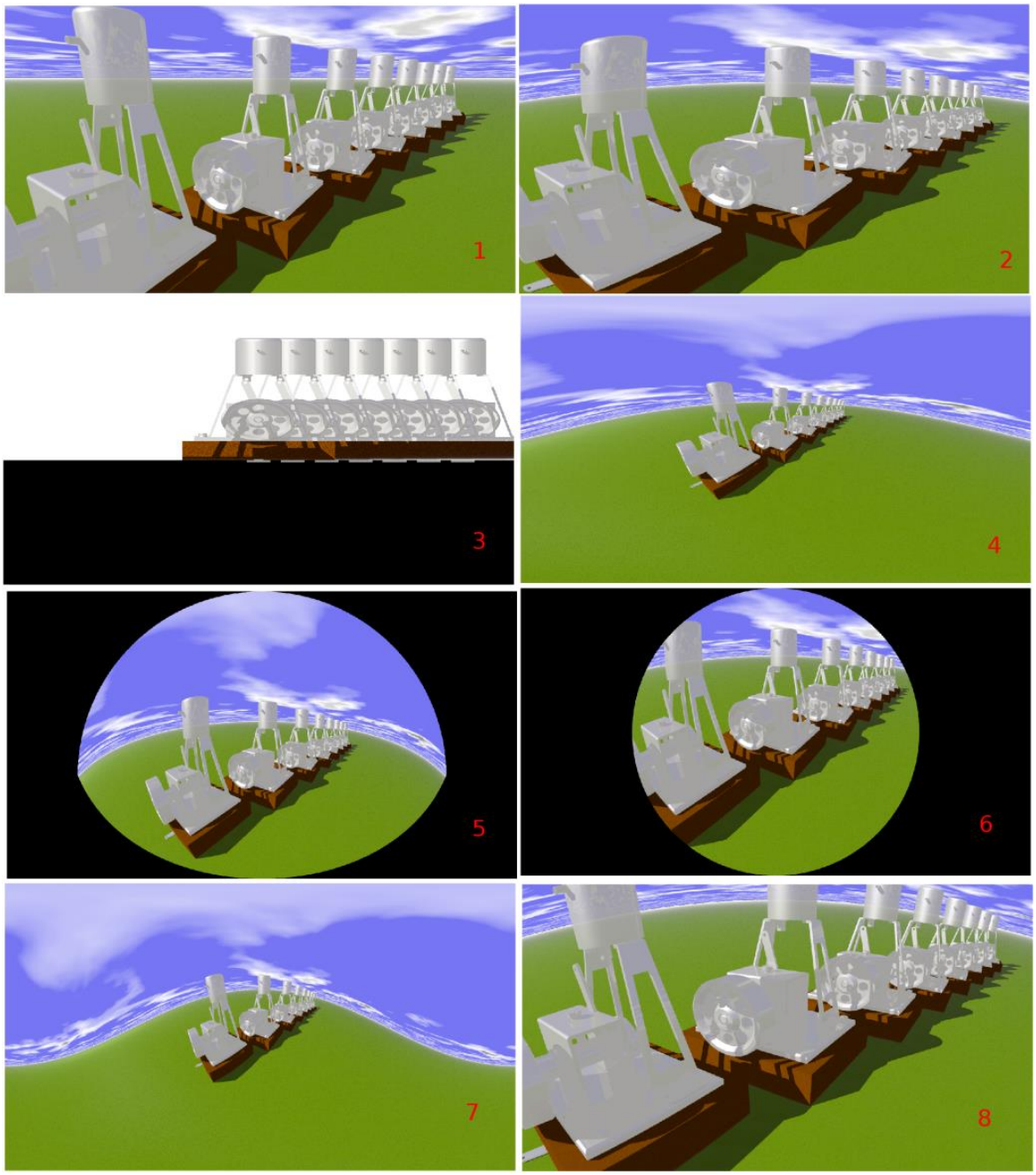
7.2 Opis podstawowych elementów definiujących scenerię

1. Definicja kamery opiera się na ustawieniu pozycji (lokation), punktu, na który jest skierowana (look at) oraz wyborze typu odwzorowania (domyślnie jest to kamera perspektywiczna). Dodatkowo, definicję można uzupełnić o symulację głębi ostrości, definiując obszar ostrego odwzorowania (focal point), stopnia rozmycia przed i za obszarem, na którym jest ustawiona ostrość (aperture), jakość rozmycia definiuje natomiast próbkowania (samples), przykładowa definicja kamer:

```
// kamera ------------------------------------------------------------
#declare kamera_xyz = camera {
location <50,50, -60> // widok skośny
look_at <0.0, 0, 0.0> //pkt., na który skierowano kamerę
focal_point <0.5, 0.5, 0.5> //pkt. ustawienia ostrości
aperture 1.5 // stopień rozmycia poza obszarem ostrości
blur_samples 20 //próbkowanie
}
#declare kamera_xy = camera { location <0,0, -60> // widok z przodu
look_at <0.0, 0, 0.0> //pkt., na który skierowano kamerę
focal_point <0.5, 0.5, 0.5> //pkt. ustawienia ostrości
aperture 1.5 // stopień rozmycia poza obszarem ostrości
blur_samples 20 //próbkowanie} //koniec df kamery ----------------------------
```

Przywołanie jednego z ustawień kamery realizowane jest komendą: camera{kamera_xy}.

W zależności od zastosowanej kamery, uzyskujemy odmienne odwzorowania, co przedstawiono na rysunku 7.2



Rysunek 7.2. Typy kamer.
1 – perspektywiczna; 2 – szerokokątna; 3 – ortograficzna; 4 – panoramiczna; 5 – omnimax (rybie oko z kątem 180 deg); 6 – rybie oko; 7 – sferyczna; 8 – cylindryczna.
Źródło: opracowanie własne.

2. Oświetlenie określane jest przy wykorzystaniu podstawowych typów źródeł światła, definiowanych opisanymi w komentarzach poniżej parametrami:
   - Polecenie utworzenia regularnego punktowego źródła światła:

```
light_source {
<0,0,0> // współrzędne lokalizacji światła
color rgb <1,2,3> // emitowany kolor w skali RGB
translate <-10, 10, -10> //wektor przemieszczeń
}
```

- polecenie generowania oświetlenia kierunkowego, dającego stożkowy snop światła

```
light_source {
<0,0,0> // współrzędne lokalizacji światła
color rgb <1,2,3> //emitowany kolor w skali RGB
spotlight // rodzaj oświetlenia
translate <40, 80, -40> // wektor przemieszczeń
point_at <0, 0, 0> // kierunek oświetlenia
radius 10 // promień oświetlenia
tightness 49 // rozmycie krawędzi w skali 1-100
falloff 12 // promień zaniku intensywności
translate <-10, 10, -10> //wektor przemieszczeń
}
```

- polecenie dodawania oświetlenia kierunkowego, dającego walcowy snop światła:

```
// create a point "spotlight" (cylindrical directed) light source
light_source {
<0,0,0> // współrzędne lokalizacji światła
color rgb <1,2,3> //emitowany kolor w skali RGB
spotlight // rodzaj oświetlenia
cylinder // wariant walcowy
translate <40, 80, -40> // wektor przemieszczeń
point_at <0, 0, 0> // kierunek oświetlenia
radius 10 // promień oświetlenia
tightness 49 // rozmycie krawędzi w skali 1-100
falloff 12 // promień zaniku intensywności
translate <-10, 10, -10> //wektor przemieszczeń
}
```

- wymuszenie równoległego rozchodzenia się fotonów poprzez dodanie w definicji oświetlenia poniższych komend:

```
parallel
point_at <10, 0, 0>
```

- oświetlenie powierzchniowe generuje siatkę źródeł światła rozmieszczonych na powierzchni:

```
light_source {
<0,0,0> // współrzędne lokalizacji światła
color rgb <1,2,3> //emitowany kolor w skali RGB
area_light // typ oświetlenia
<10, 0, 0> <0, 0, 10> // wymiary
5, 5 // ilość źródeł światła w tym przypadku 5x5=25
adaptive 0 // adaptacyjne
jitter // losowe wygładzenia krawędzi
circular // kształt
orient // orientacja
translate <-10, 10, -10> //wektor przemieszczeń
}
```

- dodanie widocznej reprezentacji źródła światła w definicji obiektu:

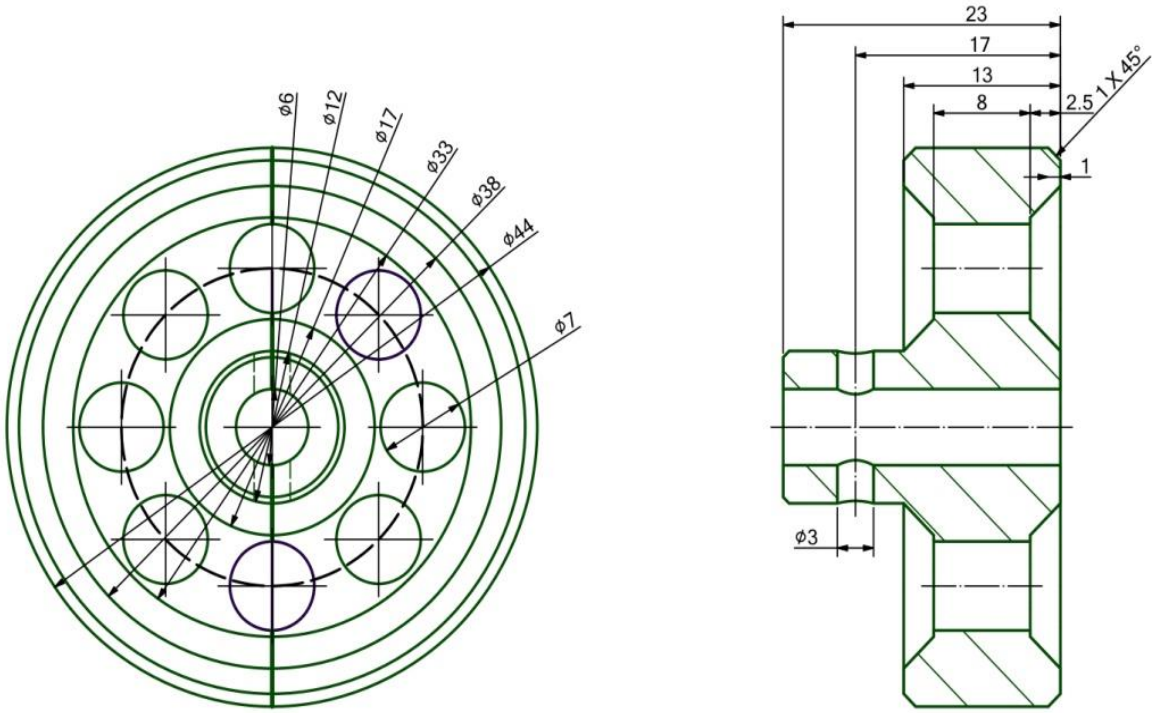
```
looks_like { sphere { 1*x, 6 pigment { Blue } } }
```

8. Tok postępowania podczas programowania scenerii w POV-Ray

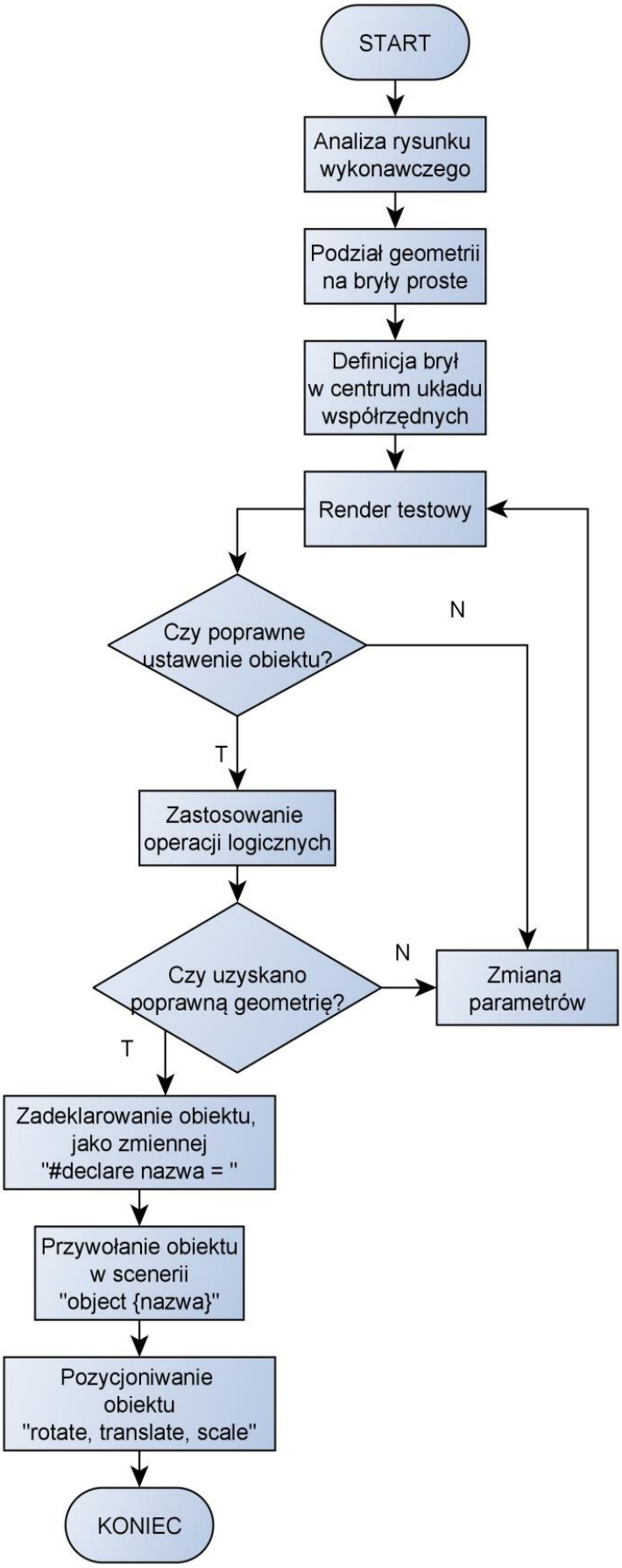
Na podstawie doświadczenia w prowadzeniu zajęć z przedmiotu „Grafika i komunikacja człowiek – komputer" zauważyć można kilka problemów, z jakimi borykają się studenci. Podstawowym wyzwaniem jest połączenie informacji płynących z kodu źródłowego z wyobrażeniem przestrzennym modelu, jaki uzyskamy, kompilując ten kod. Istotne w tym zakresie są dwa aspekty:

- zwizualizowanie sobie przestrzeni scenerii w kontekście napisanego kodu, zwłaszcza podczas zmian geometrii obiektów oraz ich położenia w przestrzeni;
- zrozumienie relacji logicznych między obiektami (CSG)

W niniejszym rozdziale szczegółowo opisano algorytm 8.2, pozwalający na uporządkowane przejście z dokumentacji technicznej (rysunku wykonawczego części) na kod źródłowy i jego kompilację. Przejście przez poszczególne kroki rozumowania pozwoli na zrozumienie przestrzeni opisywanej w kodzie źródłowym oraz stopniowe przejście od rysunku na przykładzie koła zamachowego 8.1 przez bryły proste, relacje logiczne, po kompletny kod generujący poprawną geometrię podczas renderingu.



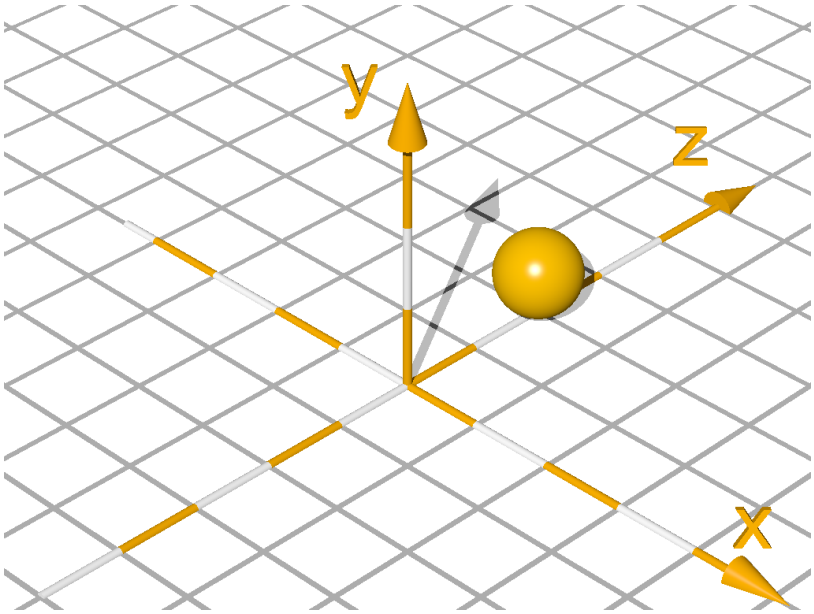
Rysunek 8.1. Koło zamachowe.
Źródło: opracowanie własne.



Rysunek 8.2. Tok postępowania przy modelowaniu.
Źródło: opracowanie własne.

8.1 Wyobraźnia przestrzenna

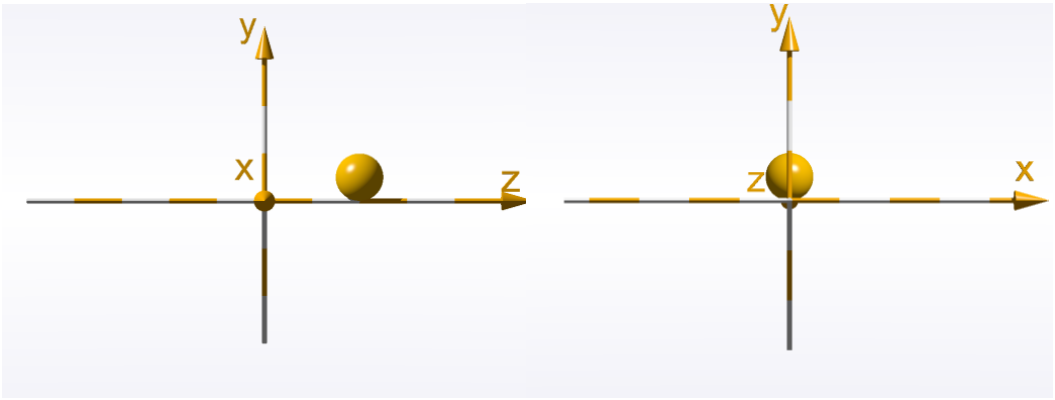
Dysponując dokumentacją techniczną, problem odniesienia geometrii dostępnej na rysunku dwuwymiarowym do jej reprezentacji w przestrzeni ogranicza się w dużej mierze jedynie do określenia, który z rzutów prostokątnych jest skorelowany z płaszczyznami XY, XZ oraz YZ w kartezjańskim układzie współrzędnych 8.3. W drugiej kolejności należy określić charakterystyczne położenia kamer ustawione na przywołane płaszczyzny i podczas modelowania przełączać się pomiędzy odpowiednimi kamerami. Ostatnim krokiem jest uświadomienie sobie, jakie w każdym z ustawień kamery mamy osie pionowe i poziome (łącznie z ich zwrotem).



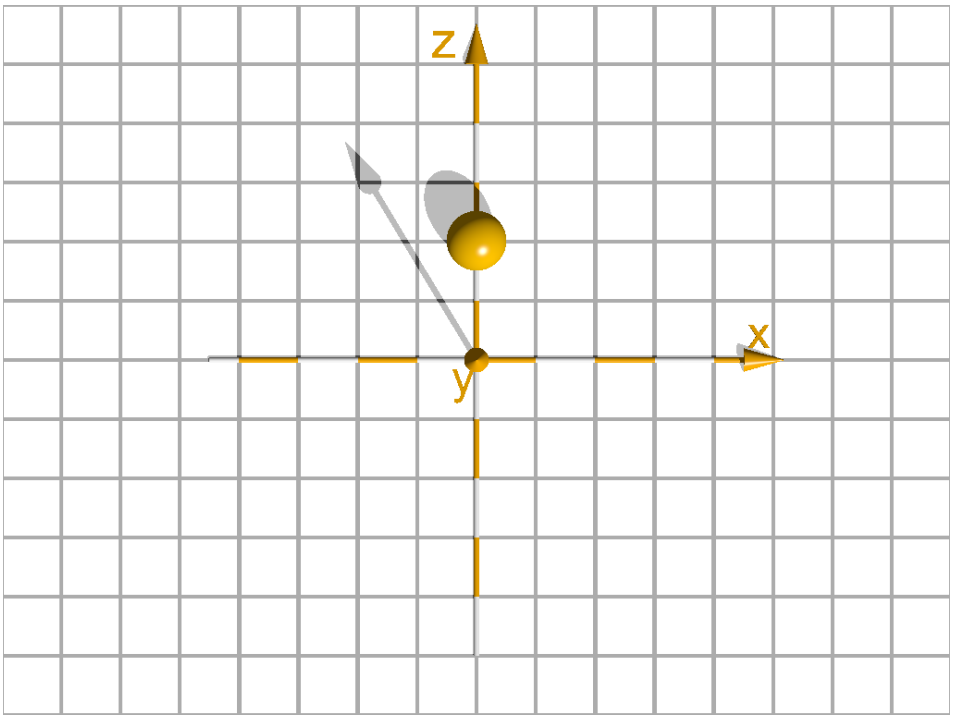
Rysunek 8.3. Kartezjański układ współrzędnych.
Źródło: opracowanie własne.

Dobrą praktyką przy definicji kamer jest sporządzenie zestawu zadeklarowanych podstawowych ustawień kamery jako zmienne, które w łatwy sposób można przywołać, zmieniając jeden parametr przed kompilacją z kamerą C_skos np. na kamerę C_xy. Poniżej przedstawiono propozycję zorganizowania deklaracji kamer, w celu usprawnienia modelowania.

```
#declare C_xy = camera { angle 15
location <0.0 , 0 ,-40.0>
look_at <0.0 , 0 , 0.0>}
#declare C_skos = camera { angle 14
location <20.0 , 20 ,-20.0>
look_at <0.0 , 1 , 0.0>}
#declare C_yz = camera { angle 14
location <50 , 0 , 0>
look_at <0.0 , 0 , 0.0>}
#declare C_xz = camera { angle 14
location <0.0 , 65 ,0>
look_at <0.0 , 0 , 0.0>}
camera{C_skos} //przywołanie zadeklarowanych ustawień kamery
```



(a) Kamera C_yz (b) Kamera C_xy



(c) Kamera C_xz

Rysunek 8.4. Widok przy poszczególnych ustawieniach kamer.
Źródło: opracowanie własne.

Podczas pierwszego kontaktu z oprogramowaniem może być ustawienie w pionie osi Y, zwłaszcza dla inżynierów, którzy zetknęli się wcześniej z programowaniem obrabiarek sterowanych numerycznie, gdzie na np. przy frezarkach pionowa oś to Z.

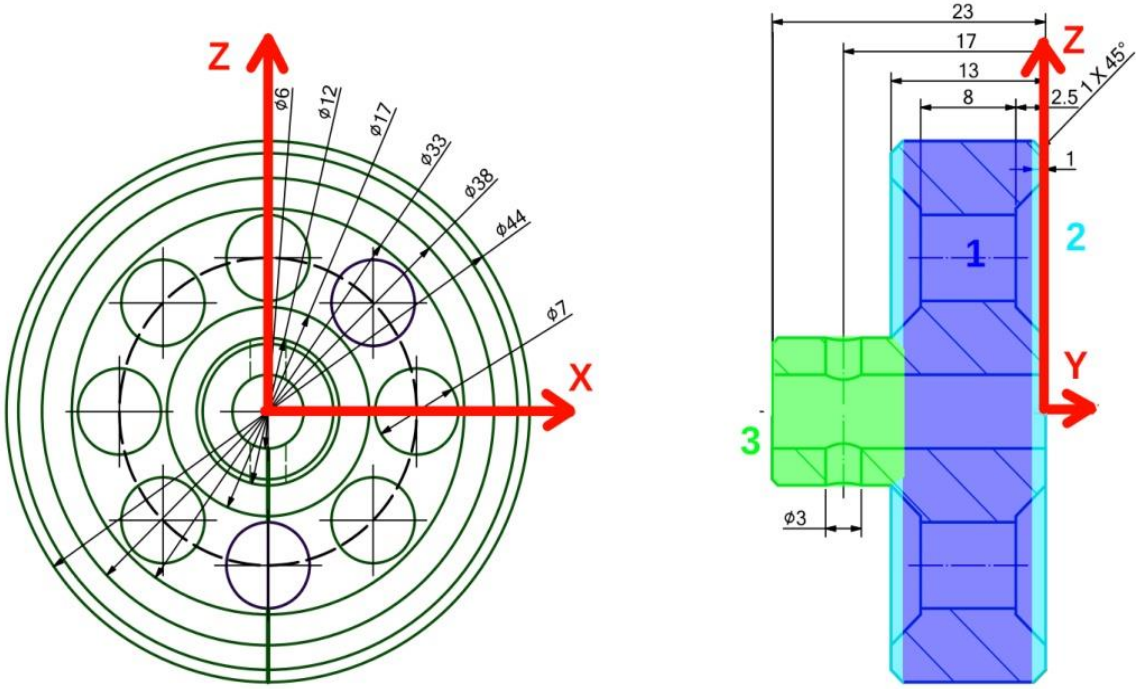
Niemniej jednak zapamiętać należy, że podstawowy układ współrzędnych ustawiony jest z osią Y skierowaną ”w niebo”, stąd ewentualna zmiana orientacji układu współrzędnych możliwa jest przy zastosowaniu komendy „sky z” lub „sky x”, co będzie skutkowało pionowym ustawieniem osi odpowiednio Z lub X. Domyślnie (bez zmiany orientacji) dodatnie współrzędne osi Z występują ”w głąb monitora”, a dodatnie wartości osi X na prawo, patrząc w płaszczyźnie XY. Ustawienie układu współrzędnych przedstawiono na rysunku 8.3, na którym znajduje się dodatkowo kula przesunięta o 2 jednostki w osi Z (współrzędna dodatnia).

8.2 Analiza rysunku

Zaproponowany algorytm postępowania przy modelowaniu 8.2 przewiduje w pierwszej kolejności analizę rysunku. Na tym etapie dwa aspekty są najważniejsze:

- orientacja poszczególnych płaszczyzn rzutów względem globalnego układu współrzędnych,
- lokalizacja obiektu względem zera układu współrzędnych (najbardziej zasadne jest skorelowanie baz pomiarowych z punktami zerowymi poszczególnych osi o ile model na to pozwala).

W przypadku koła zamachowego 8.1, w pierwsze kolejności zauważyć można, że obiekt jest osiowo-symetryczny, co zdecydowanie ułatwia modelowanie i sugeruje, że wszystkie obiekty powinny być ustawiane w centrum scenerii. Dla potrzeb niniejszego przykładu oś przedmiotu ustawiono tak, by pokrywała się z osią Y układu współrzędnych, a powierzchnia styku koła z łożyskiem spoczywała na płaszczyźnie XZ, oznaczenie osi naniesiono na rysunku 8.5.



Rysunek 8.5. Ustawienie obiektu na scenerii oraz podział na bryły proste.
Źródło: opracowanie własne.

8.3 Podział geometrii na bryły proste i ich definicja

Kolejnym krokiem jest wydzielenie z obiektu brył prostych, przy czym niekoniecznie chodzi tutaj wyłącznie o pocięcie geometrii na fragmenty, selekcjonując obiekty, które należy dodać w kolejnych fragmentach kodu źródłowego, lecz również o obiekty, które będą usuwane z modelu. Strategii możliwych do zastosowania przy każdym modelu jest wiele, a wybór najkorzystniejszej z nich zawsze opiera się o doświadczenie i oszacowanie czasu oraz wysiłku włożonego w zaprogramowanie wygenerowania obiektu.

Dla przytoczonego rysunku na początku założono, że podobnie jak w procesie rzeźbienia z obiektu o większych wymiarach (z naddatkami), zostanie usunięte ”to, co nie jest potrzebne”, czyli wycięte zostaną otwory oraz stożkowe wybrania. Na rysunku 8.5 zaznaczono obiekty z których zbudowany zostanie obiekt wyjściowy: 1 – walec, 2 – stożki, 3 – walec z zaokrąglonymi krawędziami.

Kolejność definicji brył prostych jest dowolna w tym przypadku. Zastanowienie nad kolejnością istotne jest dla operacji logicznych z wykorzystaniem polecenia „difference", przy zastosowaniu której od obiektu wprowadzonego jako pierwszy odejmowane są wszystkie pozostałe.

Dla potrzeb tego przykładu w tle pozostawiony zostaje układ współrzędnych, dla łatwiejszego orientowania się w przestrzeni scenerii. Jako pierwszy obiekt zadeklarowano stożek ścięty (rysunek 8.5 – 2). Obiekt należy przywołać poleceniem „Insert" – „Basic shapes" – „round cone truncated". Co do wymiarów, na rysunku 8.5 zawarte są one w opisie fazowania: 1×45^{o}, walec natomiast ma średnicę Ø44[mm], czyli ustawienia obiektu podsumować można jako:

- Środek pierwszej podstawy zlokalizowany jest w miejscu o współrzędnych <0, 0, 0>, a jej promień wynosi 44/2 − 1[mm], wymiar można zdefiniować działaniem matematycznym, które zostanie przeliczone podczas kompilacji.
- Środek drugiej podstawy zlokalizowany jest w miejscu o współrzędnych <0, 1, 0>, a jej promień wynosi 44/2[mm].
- Aby zachować spójność z oznaczeniami kolorystycznymi zawartymi w rysunku 8.5, ustawiono kolor obiektu jako „Cyan".

W kole zamachowym występują dwa takie elementy, więc można skopiować opisywany fragment kodu (efekt kompilacji przedstawia rysunek 8.6 a), dokonując na kopii dwóch transformacji:

Rotacji wokół osi X lub Z – rotate <180, 0, 0>
Translacji o wektor translate <0, 13, 0>
Fragment kodu opisującego pierwszy i drugi stożek ścięty:

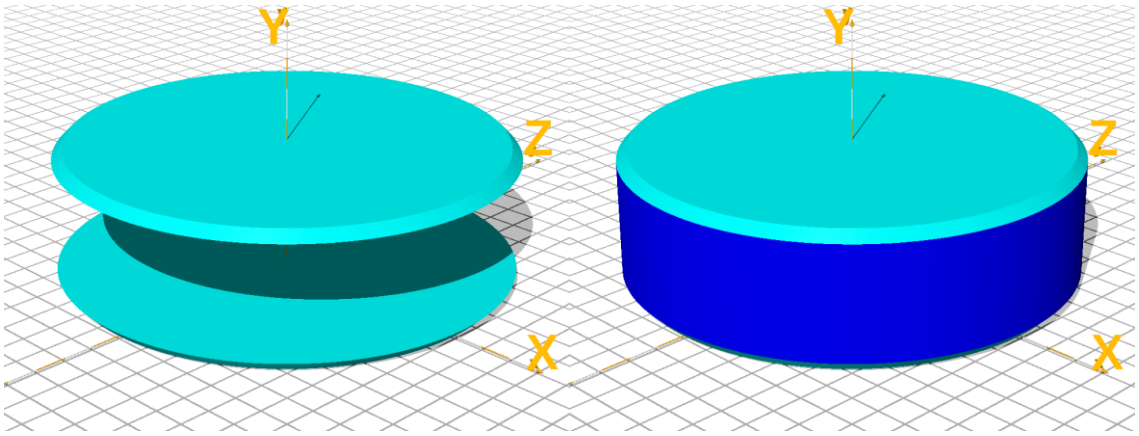
```
cone{ <0,0,0>,44/2-1,<0,1,0>,44/2
texture{ pigment{ color Cyan}
// pigment{ color rgb<1.00,0.60,0.00>}
finish { phong 0.5 reflection{ 0.00 metallic 0.00} }
} // end of texture
scale <1,1,1> rotate<0,0,0> translate<0,0,0>
} //koniec pierwszego stożka ściętego
cone{ <0,0,0>,44/2-1,<0,1,0>,44/2
texture{ pigment{ color Cyan}
// pigment{ color rgb<1.00,0.60,0.00>}
finish { phong 0.5 reflection{ 0.00 metallic 0.00} }
} // end of texture
scale <1,1,1> rotate<180,0,0> translate<0,13,0>
} // koniec drugiego stożka ściętego
```

Definicja walca oznaczonego na rysunku 8.5 jako 1 sprowadza się do przywołania walca ustawionego w osi Y, korzystając z polecenia „Insert" – „Basic shapes round" – „cylinder y". Parametry definiujące geometrię, to:

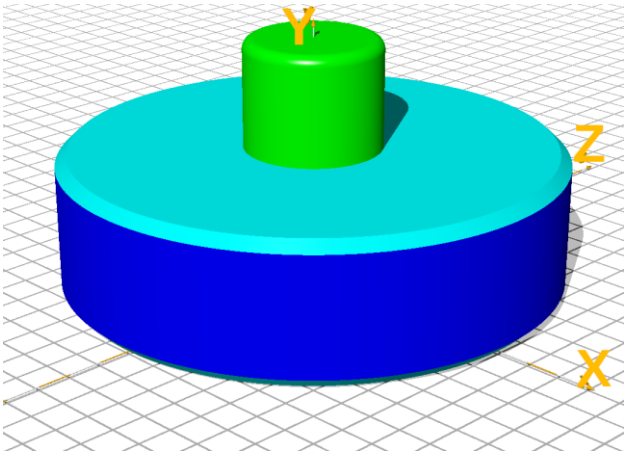
- Promień walca 44/2[mm].
- Wysokość walca definiowana współrzędnymi dwóch podstaw <0, 0, 0>, <0, 13 − 2, 0>.
- Translacja, która stanowi uwzględnienie fazowania reprezentowanego przez stożek translate <0, 1, 0>.

Definicję walca, wraz z jego wizualizacją, przedstawiono na rysunku 8.6b i w poniższym kodzie źródłowym:

```
cylinder { <0,0,0>,<0,13-2,0>, 44/2
texture { pigment { color Blue }
//normal { bumps 0.5 scale <0.005,0.25,0.005>}
finish { phong 0.5 reflection{ 0.00 metallic 0.00} }
} // end of texture
scale <1,1,1> rotate<0,0,0> translate<0,1,0>
} // koniec cylindra
```



(a) Definicja stożków ściętych – 2 (b) Definicja walca – 1



(c) Definicja walca z zaokrągleniami – 3

Rysunek 8.6. Wizualizacja poszczególnych faz modelowania koła zamachowego.
Źródło: opracowanie własne.

Definicja walca z zaokrąglonymi krawędziami, oznaczonego na rysunku 8.5 jako 3, sprowadza się do zastosowania polecenia „Insert” – „Shapes2” – „Round Cylinder”. Parametry definiujące geometrię to:

- Promień walca 12/2[mm].
- Promień zaokrąglenia krawędzi 1[mm].
- Wysokość walca definiowana dwiema podstawami <0, 0, 0>, <0, 11, 0>.
- Translacja, uwzględniająca, że jedno z zaokrągleń jest zbędne translate <0, 13−1, 0>.

Definicję walca, wraz z jego wizualizacją, przedstawiono na rysunku 8.6c i w poniższym kodzie źródłowym:

```
object { //Round_Cylinder(point A, point B, Radius, EdgeRadius, UseMerge)
Round_Cylinder(<0,0,0>, <0,11,0>, 12/2, 1, 0)
texture{ pigment{ color Green }
//normal { radial sine_wave frequency 30 scale 0.25 }
finish { phong 1 }
}
scale<1,1,1> rotate<0, 0,0> translate<0,13-1,0>
} // koniec walca zaokrąglonego
```

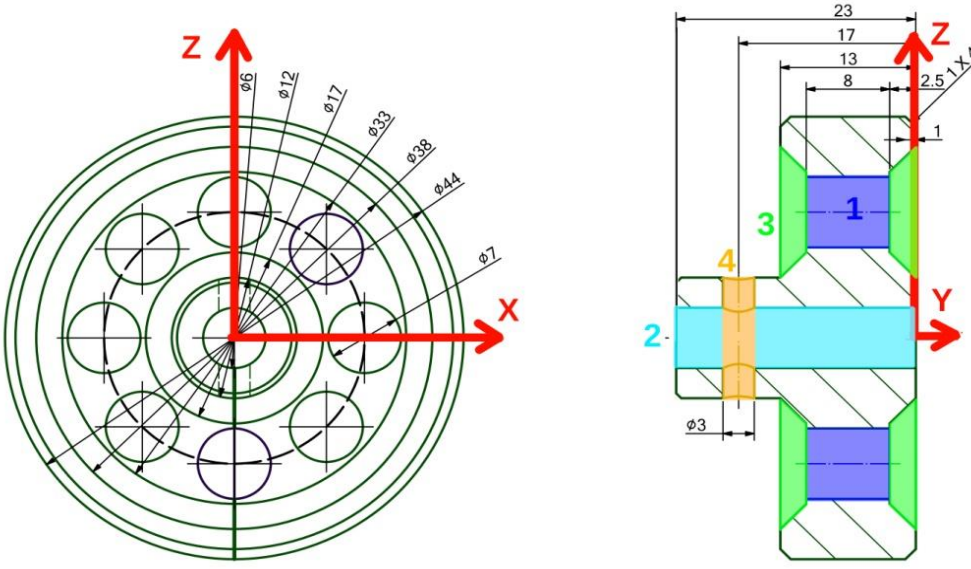
8.4 Stosowanie operacji logicznych

Do tej pory definicja koła zamachowego opierała się na ustawieniu w scenerii niezależnych (oddzielnych) obiektów, co obrazuje również różna kolorystyka poszczególnych brył. Dopiero gdy zostaną one połączone logicznie, uzyskuje się dobrej jakości model.

Dla uporządkowania dotychczasowej składni należy połączyć wszystkie trzy obiekty, tj. dwa stożki, walec i walec zaokrąglony, niejako ”obejmując” je poleceniem union, w uproszczeniu:

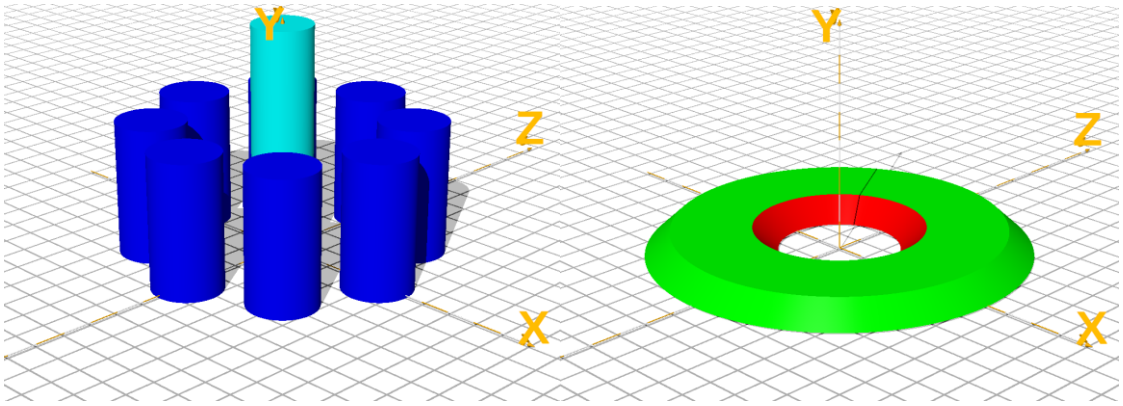
```
union{
definicja walca zaokrąglonego
definicja pierwszego stożka ściętego
definicja cylindra
definicja drugiego stożka ściętego
}
```

Pozostaje usunąć wszystkie naddatki za pomocą operacji difference. W tym przypadku, podobnie jak dla wszystkich operacji logicznych, dobrą praktyką, pozwalającą uniknąć wielu błędów podczas programowania, jest zdefiniowanie obiektów, a dopiero po uzyskaniu pewności, że wymiary i wzajemne położenie brył jest poprawne, obłożenie składni poleceniami logicznymi.

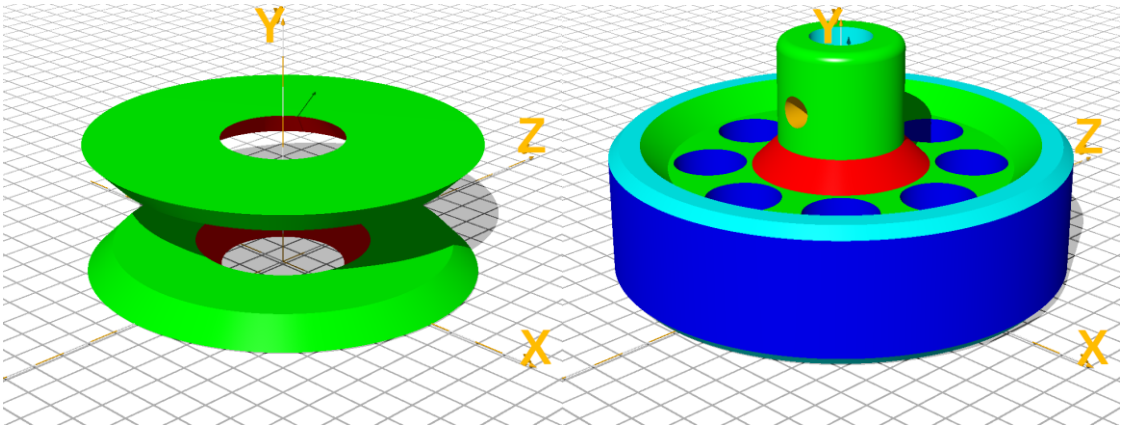


Rysunek 8.7. Obszary modelowane z wykorzystaniem operacji logicznych.
Źródło: opracowanie własne.

W rozważanym przypadku zidentyfikowano cztery obszary oznaczone na rysunku 8.7 cyframi od 1 do 4. Osiem otworów oznaczonych na rysunku 8.7, jako 1 – jest rozmieszczonych na okręgu o średnicy Ø25[mm]. Zaprogramowanie obiektów, wykorzystanych jako negatyw otworów przy operacjach logicznych, sprowadza się do zdefiniowania walca o średnicy Ø7[mm], a następnie ośmiokrotnemu przywołaniu uzyskanej geometrii na okręgu podziałowym.



(a) Otwory – szyk kołowy (b) Negatyw stożkowego wybrania materiału



(c) Dwukrotne przywołanie zmiennej negatyw_stożki (d) Ostateczny efekt po zastosowania operacji logicznych

Rysunek 8.8. Wizualizacja poszczególnych faz operacji logicznych.
Źródło: opracowanie własne.

Poniżej przytoczono fragment kodu, w którym wykorzystano pętlę logiczną, w której i oznacza numer pozycji wywołania, wykorzystany przy rotacji obiektu względem osi Y. Definicja pętli została skonstruowana tak, aby przy kolejnych iteracjach generowała $i \in \langle 0, 315\rangle$ z krokiem co 45°. Oś otworu osadzenia koła na czopie wału głównego również zorientowana jest w osi Y, więc efekt kompilacji przedstawiono na rysunku 8.8a.

```
#for (i,0,315,45) //wywołanie cylindra w szyku kołowym
cylinder{<0,-1,0><0,14,0>7/2
texture{ pigment{ color Blue}}
translate z*(25/2) rotate y*i}
#end
cylinder{<0,-1,0><0,24,0>6/2
texture{ pigment{ color Cyan}}
}
```

W przypadku stożkowych wybrań w kole zamachowym, oznaczonych jako 3 na rysunku 8.7, wymagane jest modelowanie różnicy między dwoma stożkami ściętymi. Podobnie jak podczas podziału geometrii na bryły proste – rysunek 8.6c, najpierw należy poprawnie dobrać parametry brył geometrycznych, a dopiero potem wykorzystać operacje logiczne na bryłach. Przy ustalaniu współrzędnych podstaw stożków współrzędne zostały zmienione o pomijalnie małą z punku widzenia skali obiektu wartość. Pozwala to wyeliminować pokrywające się powierzchnie pomiędzy obiektami – w innym przypadku nie jest możliwe uzyskanie poprawnych rezultatów operacji logicznych.

Parametry definiujące stożki, to:

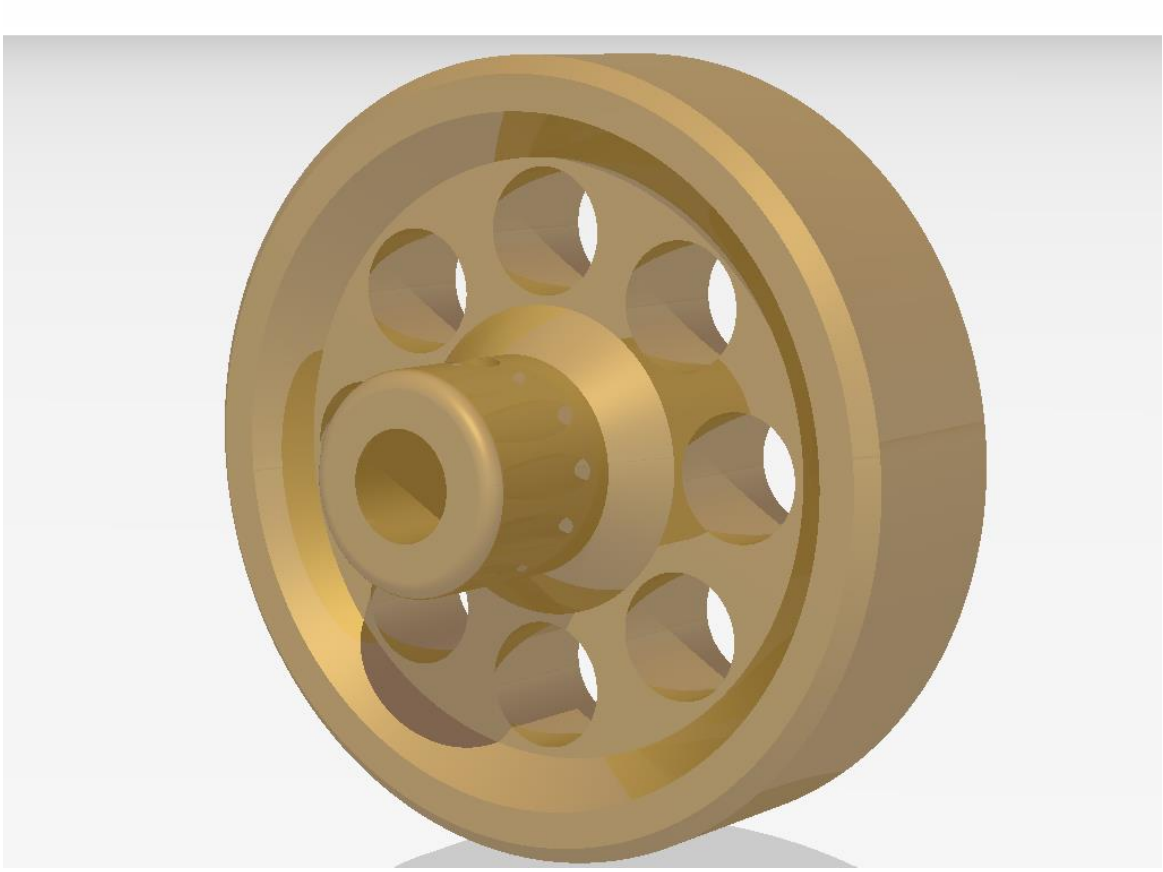
- Definicja pierwszego stożka: <0, −0.0001, 0>, 38/2, <0, 2.5001, 0>, 33/2.
- Definicja drugiego stożka: <0, −0.0002, 0>, 12/2, <0, 2.5002, 0>, 17/2.

W rozważanym przykładzie po uzyskaniu poprawnych rezultatów dwa stożki objęto operacją „difference", a następnie zadeklarowano zmienną "negatyw_stozki". W tym przypadku uzyskany kształt należy dwukrotnie wykorzystać – przywołanie zadeklarowanej zmiennej, pod którą kryje się obiekt, pozwala na utworzenie bardziej zwięzłego kodu. Drugie przywołanie zmiennej przekształcono dodatkowo poprzez obrót względem osi X o kąt 180°, a następnie wykonano translację wzdłuż osi Y o wartość 13[mm], czyli o grubość tarczy koła zamachowego.

Poniżej zamieszczono fragment kodu źródłowego, a efekt kompilacji przedstawiono na rysunku 8.8b i 8.8c.

```
#declare negatyw_stozki =
difference{
cone{ <0,-0.0001,0>,38/2,<0,2.5001,0>,33/2
texture{ pigment{ color Green}
// pigment{ color rgb<1.00,0.60,0.00>}
finish { phong 0.5 reflection{ 0.00 metallic 0.00} }
} // end of texture
scale <1,1,1> rotate<0,0,0> translate<0,0,0>
} // end of cone ---------------------------------
cone{ <0,-0.0002,0>,12/2,<0,2.5002,0>,17/2
texture{ pigment{ color Red}
// pigment{ color rgb<1.00,0.60,0.00>}
finish { phong 0.5 reflection{ 0.00 metallic 0.00} }
} // end of texture
scale <1,1,1> rotate<0,0,0> translate<0,0,0>
} // end of cone ---------------------------------
}
object{negatyw_stozki}
object{negatyw_stozki rotate x*180 translate y*13}
```

Ostatnie etap polega na wykorzystaniu polecenia „difference" w celu usunięcia z bryły podstawowej wszystkich otworów oraz stożkowych wycięć. Od pierwszego obiektu przywołanego po tej komendzie odejmowane są wszystkie następne. Po upewnieniu się, że koło zamachowe zostało poprawnie zdefiniowane (rysunek 8.8d) zadeklarowano zmienną koło zamachowe oraz usunięto robocze tekstury, wykorzystywane dla zobrazowania poszczególnych obiektów wykorzystywanych w kolejnych fazach modelowania. Ostatni etap to nadanie cech materiałowych dla gotowego obiektu oraz przywołanie go wraz z pozycjonowaniem w gotowym złożeniu. Efekt końcowy przedstawia rysunek 8.9.



Rysunek 8.9. Render ostateczny koła zamachowego.
Źródło: opracowanie własne.

```
#declare negatyw_stozki =
difference{
cone{ <0,-0.0001,0>,38/2,<0,2.5001,0>,33/2
scale <1,1,1> rotate<0,0,0> translate<0,0,0>
} // end of cone ----------------------------------
cone{ <0,-0.0002,0>,12/2,<0,2.5002,0>,17/2
scale <1,1,1> rotate<0,0,0> translate<0,0,0>
} // end of cone ----------------------------------
} //koniec deklaracji negatyw_stozki
#declare kolo_zamachowe =
difference{
union{
object { //Round_Cylinder(point A, point B, Radius, EdgeRadius, UseMerge)
Round_Cylinder(<0,0,0>, <0,11,0>, 12/2, 1, 0)
scale<1,1,1> rotate<0, 0,0> translate<0,13-1,0>
} // koniec walca zaokrąglonego
cone{ <0,0,0>,44/2-1,<0,1,0>,44/2
scale <1,1,1> rotate<0,0,0> translate<0,0,0>
} // koniec pierwszego stożka ściętego
cylinder { <0,0,0>,<0,13-2,0>, 44/2
scale <1,1,1> rotate<0,0,0> translate<0,1,0>
} // koniec cylindra
```

```
cone{ <0,0,0>,44/2-1,<0,1,0>,44/2
scale <1,1,1> rotate<180,0,0> translate<0,13,0>
} // koniec drugiego stożka ściętego
}
#for (i,0,315,45) //wywołanie cylindra w szyku kołowym
cylinder{<0,-1,0><0,14,0>7/2
translate z*(25/2) rotate y*i}
#end
cylinder{<0,-1,0><0,24,0>6/2
}
cylinder{<0,-1,0><0,13,0>3/2
rotate x*90 translate <0,17,-6>} //otwór na pin zabezpieczający
object{negatyw_stozki}
object{negatyw_stozki rotate x*180 translate y*13}
} //koniec deklaracko koło_zamachowe
object{kolo_zamachowe
rotate<90,180,0>
translate<0,24+18,25-33/2+1+37>
texture{T_Gold_1B finish { reflection 0.1 }} } //przywołanie koła zamachowego
```
wraz z nadaniem mu cech materiałowych oraz pozycjonowaniem

9. Spis rysunków

10. Bibliografia

Caudron, R., Nicq, P.A. (2015). *Blender 3D By Example*. Birmingham: Packt Publishing Ltd.

Fisher, G.C. (2012). *Blender 3D Basics Beginner's Guide*. Birmingham: Packt Publishing Ltd.

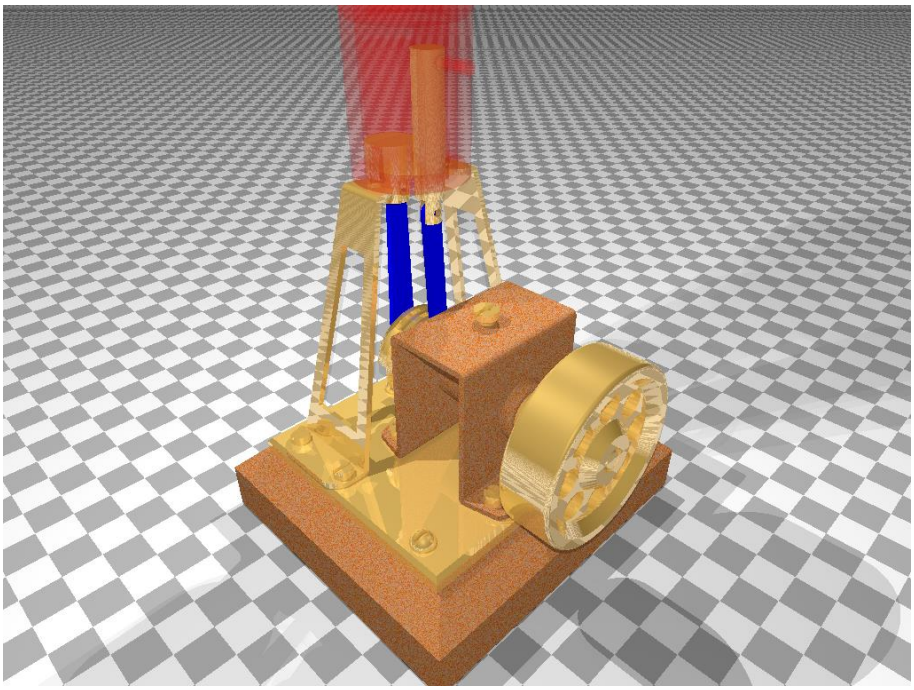
Kuklo, K., Kolmaga, J. (2007). *Blender. Kompendium*. Gliwice: Helion.

Mullen, T. (2009). *Mastering Blender*. Indianapolis: John Wiley & Sons, Inc. Pobrane z: http://www.povray.org/documentation/3.7.0/index.html.

POV-Team. (2013). *Introduction to POV-Ray*. Pobrane z: www.povray.org/documentation/index-3.6.php.

Sifre, J.C. (2014). *POV-Ray for Geometry*. Dijon: Institut Mathèmatique de Bourgogne. Pobrane z: https://cc0textures.com.

11. Załącznik – Przykład gotowej scenerii



Rysunek 11.1. Silnik parowy – efekt kompilacji kodu w POV-Ray.

```
#include "colors.inc"
#include "shapes.inc"
#include "textures.inc"
#include "golds.inc"
#include "metals.inc"
//definicja płaszczyzny na której znajduje się modelowany
//silnik
plane {
        y, -1.0
        pigment {
                checker color Gray65 color Gray30
                scale <10,10,10>
        }
}

//definicja trzech źródeł światła (współrzędne oraz kolor)
light_source { <100, 150, 100> color White }
light_source { <50, 150, -30> color White }
light_source { <-50, 150, -30> color White }
//definicja ustawienia kamery
camera {
location <100,140, 100> // układ skośny
//location <10,10, -10> // układ skośny blisko
//location <0,150,0> //widok z góry
look_at <0.0, 50, 0.0>
```

```
focal_point <0.5, 0.5, 0.5> // ustawienia ostrości 1
//focal_point < 0, 1, 0> // ustawienia ostrości 2
//focal_point < 1, 1, -6> //ustawienia ostrości 3
//aperture 0.4 //
// aperture 0.05 // cała sceneria w polu ostrości
// aperture 1.5 // rozmycie poza polem ostrości
blur_samples 4 // próbkowanie rozmycia (niska jakość)
//blur_samples 20 // próbkowanie rozmycia (wysoka
//jakość)
}
#declare woden_base1 = //formuła deklarująca obiekt
//"woden_base", jako różnicę
//difference{
        box{ //definicja prostopadłościanu
//przez podanie współrzędnych przeciwległych wierzchołków
                <-45,0,-40>, <45,20,40>
        }

        #declare fazowanie =
        prism { -50.00 ,50.00 , 4
                <-1.00, 0.00>, // pierwszy ounkt
                < 0.00, 0.00>,
                < 0.00, -1>,
                <-1.00, 0.00> // ostatni punkt=pierwszy punkt!!!
                rotate<-90,-90,0>
                scale <1, 1, 1>
                rotate<0,0,0>
                translate<0.00, 0, 0>
                } // koniec pryzmy
        #declare otwor_wkret =
        cylinder{<0,-1,0>, <0,25,0> 0.5}

        #declare woden_base = //deklaracja "woden_base",
        //jako różnicy między
        // "woden_base1" a pozostałymi elementami (obiekt
        //z usuniętymi częściami: fazowanie i otwory)
        difference{
                object{woden_base1}
                object{fazowanie scale <1, 7, 7> rotate y*-90
                        translate <-45.0001,20.0001,0> }
                //przywołany obiekt
                //"fazowanie" poddany operacją rotacji,
                //translacji oraz skalowania
                object{fazowanie scale <1, 7, 7> rotate y*90
                        translate <45.0001,20.0001,0>}
                object{fazowanie scale <1, 7, 7> rotate y*180
```

```
                translate <0,20.0001,-40.001>}
            object{fazowanie scale <1, 7, 7>
                translate <0,20.0001,40.0001>}
            object{otwor_wkret translate<-33,0,25.5-3>}
            object{otwor_wkret translate<33,0,25.5-3>}
            object{otwor_wkret translate<0,0,-25>}
            }
#declare base_plate = // definicja metalowej podstawy
//montażowej, jako różnic między prostopadłoćsianem "box",
//a fazowaniami oraz otworami montażowymi
difference{
    box {<-38,0,-64/2>,<38,4,64/2> }
    object{fazowanie rotate y*-90
        translate <-38.0001,4.0001,0> }
    object{fazowanie rotate y*90
        translate <38.0001,4.0001,0>}
    object{fazowanie rotate y*180
        translate <0,4.0001,-32.001>}
    object{fazowanie translate <0,4.0001,32.0001>}
    object{otwor_wkret scale 4 translate<-33,0,25.5-3> }
    object{otwor_wkret scale 4 translate<33,0,25.5-3>}
    object{otwor_wkret scale 4 translate<0,0,-25>}
    object{otwor_wkret scale 3 translate<-14,0,25+33-32>}
    object{otwor_wkret scale 3 translate<14,0,25+33-32>}
    object{otwor_wkret scale 3 translate<-14,0,-32+25>}
    object{otwor_wkret scale 3 translate<14,0,-32+25>}
    object{otwor_wkret scale 3 translate<-33,0,-32+8.5>}
    object{otwor_wkret scale 3 translate<33,0,-32+8.5>}
    object{otwor_wkret scale 3
        translate<-33,0,-32+18+8.5>}
    object{otwor_wkret scale 3 translate<33,0,-32+18+8.5>}
    translate y*20
}

#declare wood_screw =
union{
    difference{
        sphere{<0,0,0>3}
        box{<-3,-3,-3><3,0,3>}
        box{<-3,1,-0.5><3,3,0.5>}
        }
    cone{<0,-15,0>,0,<0,0,0>,2}
    //stożek ścięty, deklarowany współrzędnymi podstaw
    //oraz dwiema średnicami - jedna wynosi 0
    //więc zamiast stożka ściętego mamy ostre zakończenie
    //bryły
```

```
        translate y*20
}

#declare mocowanie1 =
union{
        object{wood_screw translate<-33,4.5,25.5-3>}
        object{wood_screw translate<33,4.,25.5-3>}
        object{wood_screw translate<0,4,-25>}
}

#declare CSB_holder =
difference{
        union{
                difference{
                        prism {
                                linear_sweep
                                linear_spline
                                0.0001, // początek wyciągnięcia
                                1.4999, // koniec wyciągnięcia
                                5, // ilość punktów definiujących kształt
                <-5,19>, <5,16.7>, <5,-16.7>, <-5,-19>, <-5, 19>
                        }

                        difference{
                                box{<0,0,0><5,1.5,5>translate <5-4,0,14>}
                                cylinder{<0,0,0>,<0,1.5,0>4 scale y*1.001
                                        translate <5-4,0,14>}
                        }

                        difference{
                                box{<0,0,0><5,1.5,-5>translate <5-4,0,-14>}
                                cylinder{<0,0,0>,<0,1.5,0>4 scale y*1.001
                                        translate <5-4,0,-14>}
                        }
                        translate x*(10+1.5+5)}
                difference{
                        prism {
                                linear_sweep
                                linear_spline
                                0.0001,
                                1.4999,
                                5,
                <-5,19>, <5,16.7>, <5,-16.7>, <-5,-19>, <-5, 19>
                        }
                        difference{
                                box{<0,0,0><5,1.5,5>translate <5-4,0,14>}
```

```
                cylinder{<0,0,0>,<0,1.5,0>4 scale y*1.001
                        translate <5-4,0,14>}
        }
        difference{
        box{<0,0,0><5,1.5,-5>translate <5-4,0,-14>}
        cylinder{<0,0,0>,<0,1.5,0>4 scale y*1.001
                        translate <5-4,0,-14>}
        }
        rotate y*180
        translate x*(-10-1.5-5)}
box{<-11.5,1.5,-19><-10,35,19>}
box{<11.5,1.5,-19><10,35,19>}
box{<-10,35,-19><10,36.5,19>}
intersection{
        cylinder{<0,0,0>,<0,1,0>1.5}
        box{<0,-0.0001,0><2,1.001,2> }
        scale y*38
        rotate x*-90
        rotate z*-90
        translate <-11.5,1.5,19>
}
intersection{
        cylinder{<0,0,0>,<0,1,0>1.5}
        box{<0,-0.0001,0><2,1.001,2> }
        scale y*38
        rotate x*-90
        rotate z*180
        translate <11.5,1.5,19>
}

intersection{
        cylinder{<0,0,0>,<0,1,0>1.5}
        box{<0,-0.0001,0><2,1.001,2> }
        scale y*38
        rotate x*-90
        rotate z*90
        translate <-10,35,19>
}
intersection{
        cylinder{<0,0,0>,<0,1,0>1.5}
        box{<0,-0.0001,0><2,1.001,2> }
        scale y*38
        rotate x*-90
        translate <10,35,19>
}
```

```
        }

        union{
                cylinder{<0,-0.0001,0><0,2,0>1.5}
                cylinder{<0,-0.0001,0><0,2,0>1.5 translate <33,0,28>}
                cylinder{<0,-0.0001,0><0,2,0>1.5 translate <33,0,0>}
                cylinder{<0,-0.0001,0><0,2,0>1.5 translate <0,0,28>}
                translate <-16.5,0,-14>
         }
        cylinder{<0,-0.0001,0><0,2,0>2 translate <0,35,0>}
        cylinder{<0,-0.0001,0><0,32,0>6 rotate z*90
                translate <23.5/2,18,0>}
}

#declare M3X5 =
difference{
        union{
                cylinder{<0,0,0><0,2.5,0>2.5}
                cylinder{<0,-5,0><0,0,0>1.5}
        }
        box {<-6,1.5,-0.5><6,6,0.5>}
        }
#declare mocowanie2 =
union{
        object{M3X5 translate <-16.5,0,14>}
        object{M3X5 translate <16.5,0,14>}
        object{M3X5 translate <-16.5,0,-14>}
        object{M3X5 translate <16.5,0,-14>}
}

#declare CH_bracket =
union{
        difference{ //górne mocowanie
                prism {
                        linear_sweep
                        linear_spline
                        0.0001,
                        1.4999,
                        5,
                        <0,6>, <6,6>, <6,-6>, <0,-6>, <0, 0>
                }
                difference{
                        box{<0,0,0><5,1.5,5>}
                        cylinder{<0,-0.001,0><0,1.5001,0>5 }
                        scale 1.0001 rotate y*-90
                        translate <5.0001,0,1.001>}
```

```
        difference{
            box{<0,0,0><5,1.5,5>}
            cylinder{<0,-0.001,0><0,1.5001,0>5 }
            scale 1.0001 rotate y*180
            translate <5.0001,0,-1.001>}
            cylinder{<0,-0.001,0><0,1.5001,0>1.5
            translate x*3 }
        translate y*(65.5-1.5) }
        difference{ //dolne mocowanie
        prism {
            linear_sweep
            linear_spline
            0.0001,
            1.4999,
            5,
            <0,12>, <8.8,12>, <8.8,-12>, <0,-12>, <0, 0>
            }
        difference{
            box{<0,0,0><5,1.5,5>}
            cylinder{<0,-0.001,0><0,1.5001,0>5 }
            scale 1.001 rotate y*-90
            translate <5.001,0,7.001>}
        difference{
            box{<0,0,0><5,1.5,5>}
            cylinder{<0,-0.001,0><0,1.5001,0>5 }
            scale 1.001 rotate y*180
            translate <5.001,0,-7.001>}
        cylinder{<0,-0.001,0><0,1.5001,0>1.5
            translate x*5 translate z*9}
        cylinder{<0,-0.001,0><0,1.5001,0>1.5
            translate x*5 translate z*-9}
        rotate y*180 translate x*30 }

difference{ // blacha
        prism {
            linear_sweep
            linear_spline
            0.0001,
            1.4999,
            5,
            <0,6>, <79.2-6-8.8,12>, <79.2-6-8.8,-12>,
            <0,-6>, <0, 6>
            }
        prism {
            linear_sweep
            linear_spline
```

```
                0.000,
                1.5,
                5,
                <0,6>, <81.1-6-8.8,12>, <81.1-6-8.8,-12>,
                <0,-6>, <0, 6>
                scale x*(50/(81.1-6-8.8)-0.1)
                scale z*(50/(81.1-6-8.8)-0.1)
                translate x*12
                }
        rotate z*-77.6
        translate <6,64,0> }

    intersection{ //zaokrąglona krawędź utworzona,
        //jako część wspólna między walcem,
        //a prostopadłościanem
        cylinder{<0,0,0>,<0,1,0>1.5}
        box{<0,-0.0001,0><2,1.001,2> }
        scale y*12
        rotate x*-90
        rotate z*0
        translate y*(65.5-1.5)
        translate x*6
        translate z*6
    }
    intersection{ //zaokrąglona krawędź
        cylinder{<0,0,0>,<0,1,0>1.5}
        box{<0,-0.0001,0><2,1.001,2> }
        scale y*24
        rotate x*-90
        rotate z*180
        translate y*(1.5)
        translate x*(30-8.8)
        translate z*12
    }
}

#declare bearing_end =
union{
    difference{
        union{
            cylinder{<0,0,0><0,2,0>15/2 translate x*0}
            cylinder{<0,-4,0><0,0,0>5 translate x*0}
            object { //Round_Cylinder(punkt A, punkt B,
                //średnica,     średnica fazowania, użyj
                //łączenia - definicja cylindra
                //z zaokrąglonymi krawędziami)
```

```
                Round_Cylinder(<0,0,0>, <0,3,0>, 15/2 , 1, 0)
                }
        }
        cylinder{<0,-5,0><0,5,0>3 translate x*0}
        scale<1,1,1> rotate<0, 0,0> translate<0,23+4,0> }
    //koniec 1
    difference{
        union{
            cylinder{<0,0,0><0,2,0>15/2 translate x*0}
            cylinder{<0,-4,0><0,0,0>5 translate x*0}
            object {
            Round_Cylinder(<0,0,0>, <0,3,0>, 15/2 , 1, 0)
            }
        }
        cylinder{<0,-5,0><0,5,0>3 translate x*0}
        scale<1,1,1> rotate<0, 0,180> translate<0,4,0> }
    //koniec 2
    difference{
        cylinder{<0,4,0><0,23+4,0>6 translate x*0}
        cylinder{<0,2,0><0,30,0>5 translate x*0}
        cylinder{<0,0,0><0,10,0>2 rotate z*90
            translate y*(11.5+4)}
    } //koniec tuleja
    rotate y*-90 translate y*-15.5}

#declare C = //mocowanie łożyskowania
difference{
    union{
        cylinder{<0,0,0><0,20,0>2 translate x*0}
        cylinder{<0,20,0><0,22,0>3 translate x*0}
    }
    cylinder{<0,-1,0><0,27,0>1 translate x*0}
    box{<-10,21,-0.5><10,27,0.5> }
} //koniec C mocowanie łożyskowania

#declare kolo =
difference{
    union{
        cylinder{<0,13,0><0,23,0>6}
        difference{
            object {
            Round_Cylinder(<0,0,0>, <0,13,0>, 22 , 1, 0)
            }
            cone { <0,13-2.5,0>,33/2,<0,13.001,0>,38/2
            scale <1,1,1> rotate<0,0,0> translate<0,0,0>
            } // koniec stożka
```

```
                        cone { <0,2.5,0>,33/2,<0,-0.001,0>,38/2
                        scale <1,1,1> rotate<0,0,0> translate<0,-0,0>
                        } // koniec stożka
                        }

                cone { <0,13-2.5,0>,17/2,<0,13.001,0>,12/2
                } // koniec stożka
                cone { <0,2.5,0>,17/2,<0,0,0>,12/2
                } // koniec stożka
                }
        cylinder{<0,-1,0><0,30,0>3}
        # for (i,0,315,45) //wywołanie cylindra w szyku kołowym
        cylinder{<0,-1,0><0,30,0>3.5
                translate z*(25/2) rotate y*i}
        #end
        cylinder{<0,0,0><0,30,0>1.5 rotate z*90
                translate <8,17,0>}
        rotate x*0
        } //koniec deklaracji koła zamachowego
#declare oska =
difference{
        union{
                cone { <0,0,0>,2,<0,1,0>,3}
                cylinder { <0,1,0>,<0,52,0>,3}
                cylinder { <0,52,0>,<0,60,0>,2.5}
                cone { <0,60,0>,2.5,<0,61,0>,1.5}
                }
        cylinder { <0,0,0>,<0,20,0>,1.5 rotate <0,0,90>
                translate<15,17,0>}

} // koniec deklaracji wału głównego

#declare eccentric = // definicja pierścienia z
//niecentrycznym osadzeniem (wykorbienie wału)
difference{
        cylinder{<0,0,0><0,5,0>11/2}
        cylinder{<0,-1,0><0,6,0>5/2 translate x*2.5 }
        cylinder{<0,-1,0><0,6,0>2.5/2 rotate z*90
                translate <2.5,2.5,0> }
        translate x*-2.5} //koniec eccentric

#declare cranc_spacer = // pierścień dystansowy
difference{
        cylinder{<0,0,0><0,6.5,0>7/2}
        cylinder{<0,-1,0><0,7,0>5/2 }
        translate x*0} //koniec pierścienia dystansowego
```

```
#declare cranc_pin = // śruba blokująca
union{
        cylinder{<0,0,0><0,13,0>5/2}
        translate x*0} //koniec śruby blokującej

#declare pin2x6 =
union{
        cylinder{<0,0,0><0,-6,0>2/2}
        difference{
                sphere{<0,0,0>2 }
                cylinder{ <0,0,0><0,-3,0>3}
        }
        translate y*3.0} //koniec pin2x6

#declare crank_plate = //definicja przeciwwagi
difference{
        union{
                cylinder{<0,1,0><0,2.0,0>25/2}
                cone {<0,2.0,0>25/2<0,3,0>24/2}
                cone {<0,1.0,0>25/2<0,0,0>24/2}
                object{pin2x6 translate <-8.6,3,4>}
                object{pin2x6 translate <8.6,3,4>}
                object{pin2x6 translate <0,3,4+5.5>}
        }
        cylinder{<0,-1,0><0,4,0>5/2}
        cylinder{<0,-1,0><0,4,0>5/2 translate z*-8}
        cylinder{<0,-1,0><0,4,0>2/2 translate <-8.6,0,4>}
        cylinder{<0,-1,0><0,4,0>2/2 translate <8.6,0,4>}
        cylinder{<0,-1,0><0,4,0>2/2 translate <0,0,4+5.5>}
        translate y*0} //koniec definicji przeciwwagi

#declare crank_plate1 =
union{
        difference{
                union{
                        cylinder{<0,1,0><0,2.0,0>25/2}
                        cone {<0,2.0,0>25/2<0,3,0>24/2}
                        cone {<0,1.0,0>25/2<0,0,0>24/2}
                }
                prism {
                        linear_sweep
                        linear_spline
                        -0.0001,
                        5,
                        9,
```

```
        <-25/2,1>, <-25/2+1,0>, <-2.5,4>, <2.5,4>, <25/2-1, 0>,
                <25/2, 1>, <25/2, -25/2>,<-25/2, -25/2>,<-25/2, 1>

                }

                cylinder{<0,-1,0><0,4,0>2/2 translate <-8.6,0,4>}
                cylinder{<0,-1,0><0,4,0>2/2 translate <8.6,0,4>}
                cylinder{<0,-1,0><0,4,0>2/2 translate <0,0,4+5.5>}

        }
}

#declare cranc_cale= // wykonanie złożenia z wcześniej
//zadeklarowanych obiektów
union{
        object{cranc_spacer translate <0,3,-8>}
        object{cranc_pin translate <0,0,-8>}
        object{crank_plate1 translate <0,3,0>}
        object{crank_plate}
} // koniec cranc_cale

#declare cylinder1 =
union{
        difference{
                object {
                        Round_Cylinder(<0,-3,0>, <0,38,0>, 27/2, 3, 0)
                }
                cylinder{<0,-4,0>,<0,0,0>,(27/2+1)}
                cylinder{<0,-1,0>,<0,27,0>,(12/2) translate x*-6}
                cylinder{<0,-1,0>,<0,32,0>,(7/2) translate x*7}
                cylinder{<0,-1,0>,<0,32,0>,(2/2) rotate z*-90
                        translate x*-10         translate y*25}
                cylinder{<0,-1,0>,<0,32,0>,(3/2) rotate z*-90
                        translate x*7 translate y*25}
                cylinder{<0,-1,0>,<0,8,0>,(3/2) translate z*-10}
                cylinder{<0,-1,0>,<0,8,0>,(3/2) translate z*10}
                }
        difference{
                cylinder{<0,-1,0>,<0,8,0>,(3/2) rotate z*-90
                        translate x*10 translate y*25}
                cylinder{<0,-1,0>,<0,32,0>,(2/2) rotate z*-90
                        translate x*7 translate y*25}
        }
        } //koniec cylinder1

#declare mocowanie3 =
```

```
union{
        object{M3X5 translate <-33,0,-(32-8.5)>}
        object{M3X5 translate <-33,0,-(32-8.5-18)>}
        object{M3X5 translate <33,0,-(32-8.5)>}
        object{M3X5 translate <33,0,-(32-8.5-18)>}
}// koniec mocowanie 3

#declare valve_con_rod =
difference{
        union{
                cylinder{<0,-5/2,0>,<0,5/2,0>,14/2}
                box{<-5/2,-1.5/2,0><5/2,1.5/2,40>}
                cylinder{<0,-1.5/2,40>,<0,1.5/2,40>,5/2}
        }
        cylinder{<0,-5,0>,<0,5,0>,11/2}
        cylinder{<0,-1.5,40>,<0,1.5,40>,2/2}
}

#declare piston_con_rod =
difference{
        union{
                cylinder{<0,-1.5/2,0>,<0,1.5/2,0>,8/2}
                cylinder{<0,-1.5/2,51>,<0,1.5/2,51>,5/2}
                prism {
                        linear_sweep
                        linear_spline
                        -1.5/2,
                        1.5/2,
                        5,
                        <-4,0>, <-2.5,51>, <2.5,51>, <4,0>, <-4, 0>
                        }
        }
        cylinder{<0,-5,0>,<0,5,0>,5/2}
        cylinder{<0,-1.5,51>,<0,1.5,51>,2/2}
}

#declare tlok =
difference{
        union{
                cylinder{<0,-2.5,0>,<0,6-2.5,0>,8/2}
                cylinder{<0,6-2.5,0>,<0,8+6-2.5,0>,12/2}
        }
        box{<-10,-3,-1.5/2><10,6-2.5,1.5/2> rotate y*90}
        cylinder{<0,-5,0>,<0,5,0>,2/2 rotate z*90 translate y*0}
        }
```

```
#declare zawor =
difference{
        union{
                cylinder{<0,-2.5,0>,<0,6-2.5,0>,5/2}
                cylinder{<0,6-2.5,0>,<0,31+6-2.5,0>,7/2}
        }
        box{<-10,-3,-1.5/2><10,6-2.5,1.5/2> rotate y*90}
        cylinder{<0,-5,0>,<0,5,0>,2/2 rotate z*90 translate y*0}
        cylinder{<0,-5,0>,<0,25.5+6+2.5-2.5,0>,2.5/2 rotate z*0
                translate y*0}
        box{<0,0,0><4,2.5,2.5> rotate y*0 translate
                <0,25.5+6-2.5-2.5/2,-2.5/2>}
        cylinder{<0,0,0>,<0,5,0>,2.5/2 rotate z*-90 translate
                y*(25.5+6-2.5+2.5/2)}
        cylinder{<0,0,0>,<0,5,0>,2.5/2 rotate z*-90 translate
                y*(25.5+6-2.5+2.5/2-2.5)}
}

//przywołanie wszystkich zadeklarowanych obiektów łącznie
//z transformacjami:
object{zawor texture{T_Gold_1B } rotate y*90
        translate <0,24+65.5-5-2,-64/2+17.5+6>}
object{tlok texture{T_Gold_1B } rotate y*90
        translate <0,24+65.5-5,-64/2+17.5-7>}
object{piston_con_rod rotate <-90,0,0 >
        translate <0,24+18-8,-11.5-3-8+2> texture{pigment{Blue}}}
object{mocowanie3 translate y*25.5 texture{T_Gold_1B } }
object{cylinder1 rotate y*-90
        translate<0,24+65.5,-32+8.5+18/2>
        texture{Rust pigment{color Red transmit 0.8}} }
object{cranc_cale rotate x*-90
        translate<0,24+18,25-33/2+1+37-52-5>
        texture{T_Gold_1B }}
object{eccentric rotate x*-90
        translate<0,24+18,25-33/2+1+37-52>
        texture{T_Gold_1B }}
object{oska rotate x*-90 translate<0,24+18,25-33/2+1+37>
        texture{T_Gold_1B }}
object{kolo rotate<90,180,0>
        translate<0,24+18,25-33/2+1+37>
        texture{T_Gold_1B }}
object{C translate<0,24+18,25-33/2+1> texture{T_Gold_1B }}
object{bearing_end rotate<90,0,0>
        translate<0,24+18,25-33/2+1>
        texture{Rust}}
object{CH_bracket translate <33-30+5,24,-32+18/2+8.5>
```

```
        texture{T_Gold_1B }}
object{CH_bracket rotate y*180
        translate <-(33-30+5),24,-32+18/2+8.5>
        texture{T_Gold_1B }}
object{mocowanie2 rotate y*90 translate y*(24+1.5)
        translate z*(-32+25+(33/2))
        translate x*-0 texture{T_Gold_1B }}
object{CSB_holder rotate y*90 translate y*24
        translate z*(-32+25+(33/2))
        texture{Rust}}
object{mocowanie1 texture{T_Gold_1B }}
object{woden_base texture{Rust}}
object{base_plate texture{T_Gold_1B }}
object{valve_con_rod rotate <-90,0,-3.5 >
        translate <-2.5,24+18,-11.5-3+6>
        texture{pigment{Blue}}}
```