Laboratorium Grafiki 3D i Wizualizacji Komputerowej		
Projekt I – Stół z kośćmi i sakiewką		
Krzysztof Krawulski Krzysztof Sommerrey	2015-11-22	AiR mgr. Sem. II rok II Grupa KSS

1.Cele projektu

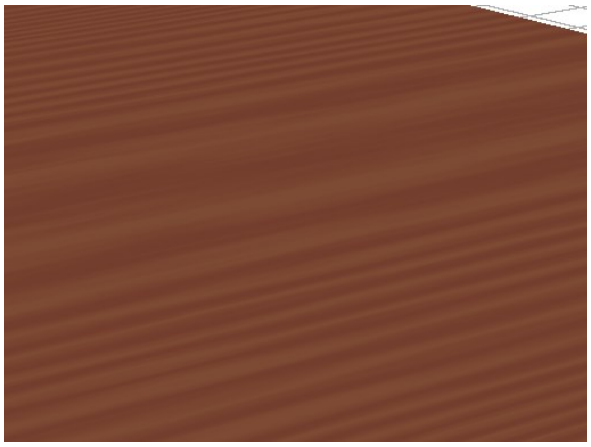
Celem projektu było stworzenie w środowisku POV – Ray sceny przedstawiającej zestaw różnych kości do gry oraz woreczka do ich przechowywania. Kości, zaczynając od k4 a kończąc na k100, miały zostać rozrzucone na stole a przy nich winna być przedstawiona sakiewka. Wykonane zostały wszystkie założenia projektowe.

2.Opis funkcji

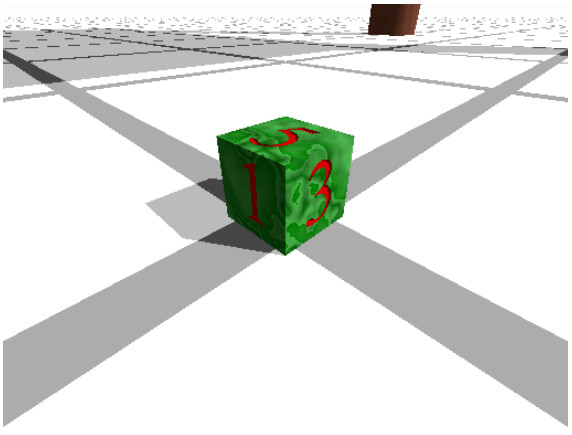
Dla zwiększenia czytelności kodu, program został podzielony na poszczególne sekcje w zależności od wykonywanych w nim operacji. Opis poszczególnych sekcji znajduje się poniżej:

2.1 Sekcja KAMERY

W tej sekcji zostały utworzone deklaracje różnych ustawień kamer w celu ułatwienia oglądania obiektów z różnych perspektyw. Dzięki takiemu rozwiązaniu, możliwe było dostosowanie położenia kamery w zależności od potrzeb. Przykładowo kamera *„Camera_1”* umożliwiała oglądanie kości na stole, natomiast „Camera_*3”* ustawiona była na początek układu współrzędnych, co umożliwiało łatwe tworzenie obiektów przed wykonaniem na nich koniecznej do stworzenia finalnej sceny translacji.



Rys 1. Widok z „Camera_1”



Rys 2. Widok z „Camera_3”

2.2 Sekcja TEKSTURY

W tej sekcji znajdują się utworzone przez nas, bądź zaczerpnięte z bibliotek programu tekstury wykorzystywane do pokrywania naszych obiektów. Deklaracje tekstur umożliwiają uzyskanie większej czytelności kodu poprzez wykorzystanie predefiniowanych nazw zamiast ponownego pisania kodu przy każdym wykorzystaniu tekstury.

2.2.1 Tekstury wbudowane

Poniżej znajdują się deklaracje oparte na zaimplementowanych w POV-Rayu teksturach. Dla przykładu:

```
#declare szklo_zielone=texture{
                                NBwinebottle
                                pigment {rgb<0.1,1,0.1>}
                                }
```

Rys 3. Tekstura imitująca zielone szkło

```
#declare granit_czarny=texture{
                              T_Grnt15
                              finish { phong 0.5 }
                              scale 1
                              }
```

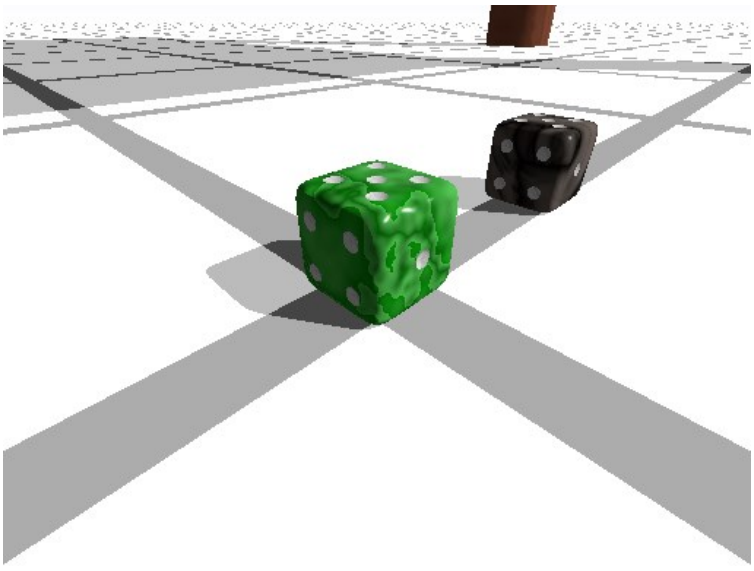
Rys 4. Tekstura imitująca czarny granit

Dzięki temu, przy późniejszym tworzeniu obiektów możemy w prosty sposób ustawić pożądaną teksturę z wykorzystaniem polecenia *„texture {nazwa_tekstury}”:*

```
object {kostka_6 texture {granit_zielony}}
object {kostka_6 texture {granit_czarny} translate <0.3,0,0>}
```

Rys 5. Utworzenie dwóch obiektów z różnymi teksturami

Takie wywołanie spowoduje wyświetlenie następującej sceny:



Rys 6. Kości o różnych teksturach

2.2.2 Tekstury wykorzystujące pliki zewnętrzne

Stworzenie tekstury woreczka do kości wymagało od nas skorzystania z rozwiązań bardziej skomplikowanych niż użycie gotowych tekstur dołączonych do programu. Aby stworzyć odpowiednią fakturę tkaniny, konieczne było utworzenie trójwymiarowego obiektu, na który został później nałożony kolor. W pliku *„fabricbumps.pov”* znajduje się kod funkcji odpowiedzialny za tworzenie tej struktury:

```
#declare redpaint = texture { pigment { color rgb <1,0,0> } finish { ambient 0.5 } }
#declare heightgrad = texture {
  pigment {
    gradient y
    pigment_map {
      [0 color Black]
      [0.5 color Black]
      [1 color White]
    }
    scale grad_h*y translate grad_off*y
  }
  finish { ambient 1 }
}

background { color Black }

difference {
  torus { 2 threadthick rotate 90*x }
  box { <-5,-5,-5> <5,0,5> texture { redpaint } }
  translate <-2,0,-2>
  texture { heightgrad }
}
difference {
  torus { 2 threadthick rotate 90*z }
  box { <-5,0,-5> <5,5,5> texture { redpaint } }
  translate <-2,0,-2>
  texture { heightgrad }
}
```

Rys 7. Fragment kodu pliku „fabricbumps.pov”

Zadeklarowanie dwóch tekstur służyło do utworzenia skali szarości w zależności od poziomu wybrzuszenia tkaniny (kolor tekstury *„redpaint”* nie ma znaczenia, gdyż jest on tylko odniesieniem skalowanym przez teksturę *„heightgrad”)*

W dalszej części kodu widzimy tworzenie poszczególnych wybrzuszeń tkaniny poprzez wykonanie operacji różnicy zbiorów na figurach *„box”* (prostopadłościan) oraz *„torus”* (pierścień o kolistym przekroju), które pokrywane są wcześniej utworzoną teksturą. Złożenie dwunastu takich operacji umożliwiło stworzenie następującego efektu:

Rys 8. Tekstura fabricbumps

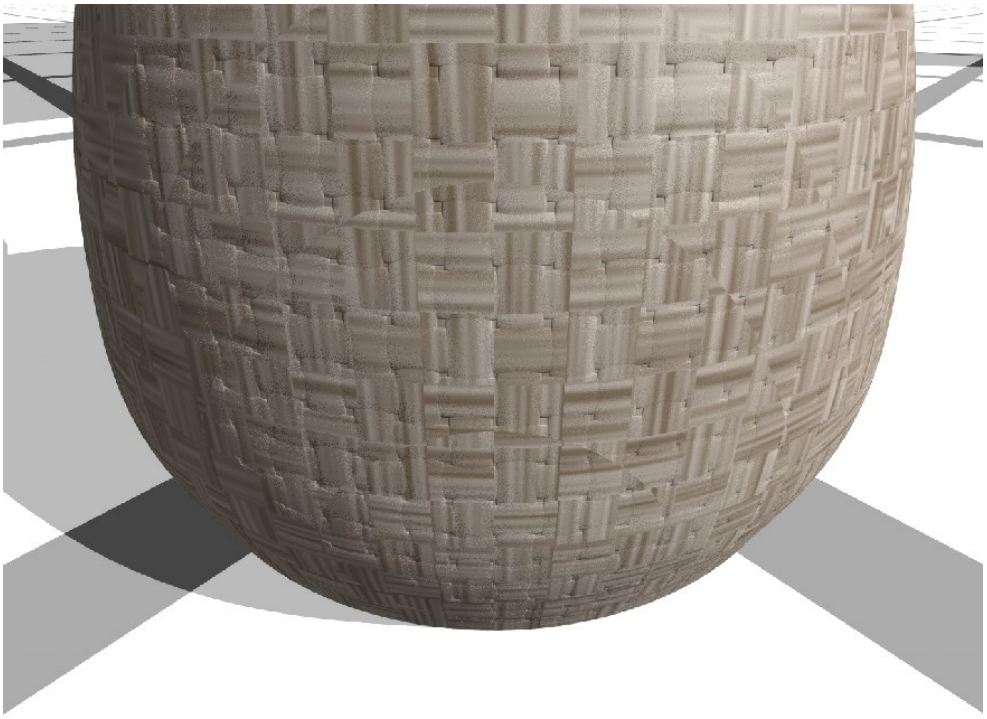
Utworzony w ten sposób plik wyjściowy mógł być potem wykorzystany w kodzie naszej sceny poprzez użycie polecenia *bump_map:*

```
//tekstury worka
#declare H_streaks = pigment { bozo color_map { [0.2 color rgb <0.4,0.3,0.2>] [0.8 color rgb <0.9,0.8,0.7>] } scale <10,0.1,1> }
#declare V_streaks = pigment { bozo color_map { [0.2 color rgb <0.4,0.3,0.2>] [0.8 color rgb <0.9,0.8,0.7>] } scale <0.1,10,1> }

#declare fabric = texture {
  pigment { checker pigment { H_streaks } pigment { V_streaks } scale 0.5*0.07 }
 normal { bump_map { png "./fabricbumps.png" bump_size 50 } scale 0.05 }
  finish { diffuse 0.8 specular 0.2 }
}
```

Rys 9. Kod tekstury worka „fabric"

Zadeklarowane na początku zmienne pigmentu *„H_streaks"* oraz *„V_streaks"* wykorzystują polecenie *bozo color_map,* które generuje losowy szum wartości wektora koloru z pomiędzy określonych wartości, aby nadać tkaninie realistyczny wygląd.

Ostateczny wygląd tkaniny uzyskujemy poprzez wykorzystanie rodzaju pigmentu *checker* (szachownica) z utworzonych wcześniej map kolorów. Poleceniem *normal* określane jest sposób, w jaki światło ma się odbijać od powierzchni obiektu, zatem dzięki wspomnianemu wcześniej *bump_map* możemy w pewien sposób „owinąć" trójwymiarową teksturę na dwuwymiarowej powierzchni tekstury. Atrybutem *bump_size* określamy wielkość wypukłości nadanych teksturze. Do poprawnego działania programu konieczne jest wcześniejsze skompilowanie pliku *„fabricbumps.pov"* w celu wygenerowania pliku *„fabricbumps.png".*



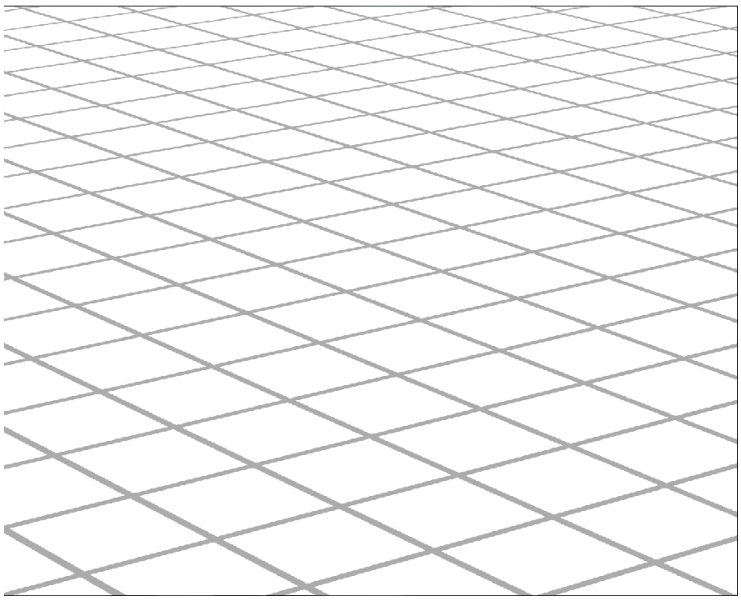
Rys 10.Wygląd tekstury na woreczku. Można wyraźnie zauważyć wybrzuszenia tekstury.

2.3 Sekcja ŚWIATŁO I SCENA

W tej sekcji znajdują się sceny wykorzystywane przez nas zarówno do tworzenia obiektów jak i zrenderowania ostatecznej wersji. Powodem zastosowania tego rozwiązania była wygoda wynikająca z naniesienia siatki układu współrzędnych do wygodnego tworzenia obiektów jak również skrócenia czasu renderowania podczas początkowych prób.

2.3.1 Scena z płaszczyzną XZ

Scena pierwsza zawierała jedynie kod nanoszący szarą podziałkę na płaszczyznę XZ, oraz pojedyncze źródło światła aby umożliwić obserwację cieni na obiekcie. Kolorystyka utrzymana w kolorze białym:



Rys 11.Wygląd pierwszej sceny

2.3.2 Scena gotowego pokoju:

Utworzenie sceny wykorzystanej w ostatecznej wersji wymagało utworzenia bryły pokoju z wyciętym oknem, aby zapewnić dodatkowe źródło światła padającego na stół.

Bryła pokoju została utworzona przez wykonanie różnicy zbiorów na dwóch prostopadłościanach, w celu nadania pewnej grubości ścianie, po czym wycięty został otwór na okno. Wykorzystana została tekstura imitująca wygląd tapety. Następnym etapem było wstawienie okiennicy oraz nadanie jej drewnianej tekstury.

```
{ difference
//bryla pokoju
  { box { <-11, 0, -11>, <16,14, 10.5> }
    box { <-10, 1, -10>, <15, 13, 10> }


    //wyciecie okna
    box { <-4, 5, 9.9>, <2, 10, 10.6> }
      texture{ pigment{ color rgb<0.76,0.75,1>}
               normal { bozo 8.5 scale 0.050 }
                finish { phong 1 reflection{ 0.05 } }
              } // end of texture

  }
  //krzyz w oknie
  box { <-1.25, 5, 10>, <-0.75, 10, 10.5> texture{ drewno }}

  box { <-4, 7.25, 10>, <2, 8, 10.5> texture{ drewno }}
  pigment { rgb 1 }
}
```

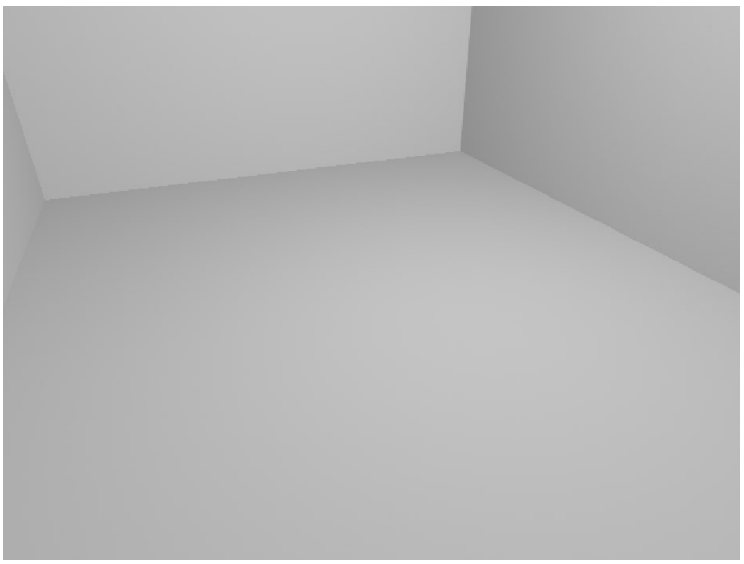
Rys 12. Kod odpowiedzialny za stworzenie bryły pokoju

Aby stworzyć iluzję światłą słonecznego zastosowana została funkcja *scattering* służąca do rozproszenia światła. Operacja zostaje wykonana we wnętrzu zdefiniowanego prostopadłościanu, aby popranie określić zakres rozproszenia:

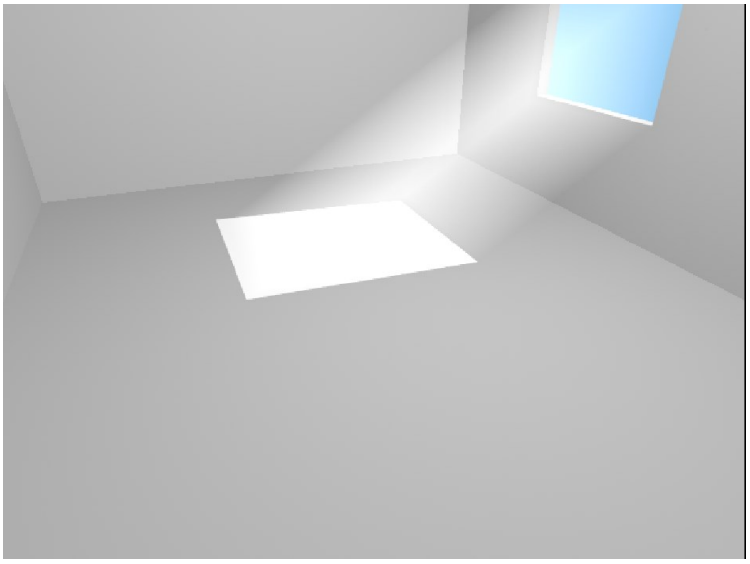
```
//ulepszenie światła zza okna
box
{ <-5, 0, -10.5>, <3, 12.5, 10.25>
  pigment { rgbt 1 } hollow
  interior
  { media
    { scattering { 1, 0.07 extinction 0.01 }
      samples 30
    }
  }
}
```

Rys 13. Rozproszenie światła zza okna w celu imitacji światła słonecznego

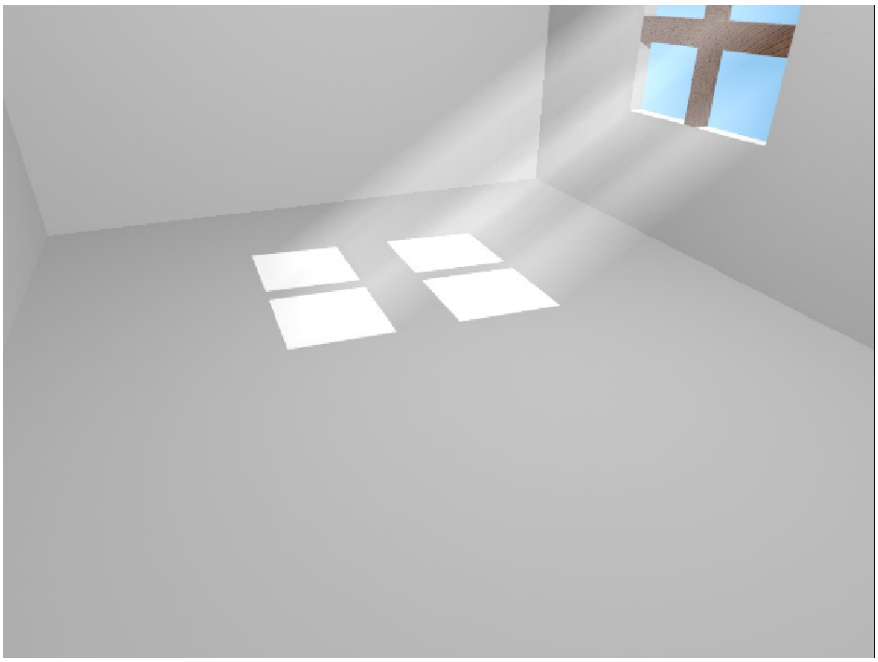
Poniżej znajdują się obrazy przedstawiające poszczególne etapy tworzenia sceny:



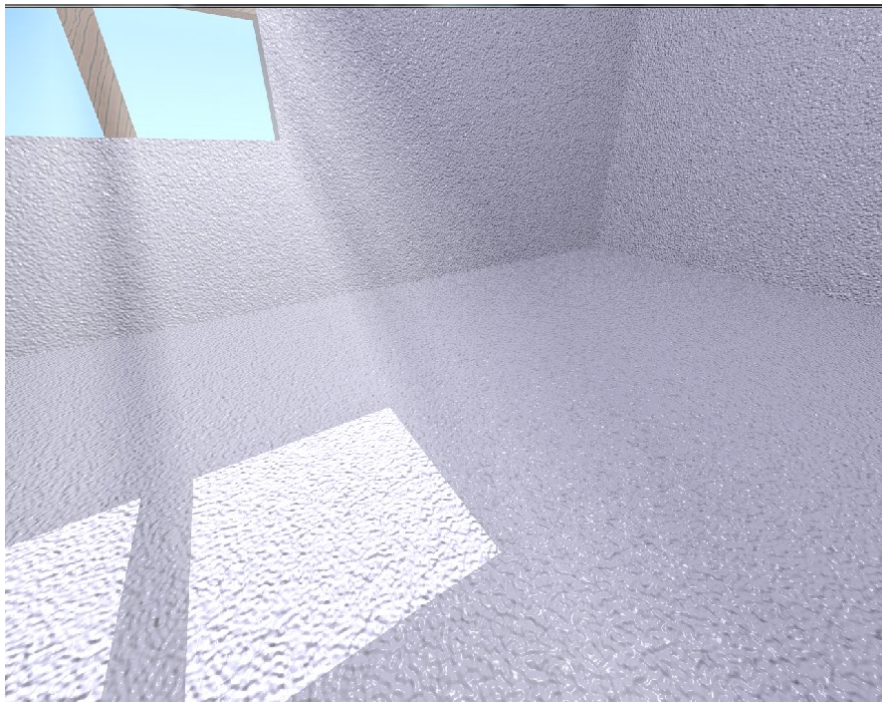
Rys 14. Bryła pokoju



Rys 15. Bryła pokoju z wyciętym oknem. Można zauważyć efekt działania rozproszenia światła.



Rys 16. Wstawienie okiennicy



Rys 17. Nadanie tekstury pokoju

2.4 Sekcja DEKLARACJE OBIEKTÓW

W tej sekcji znajdują się deklaracje poszczególnych obiektów będących elementami poszczególnych figur przed ich końcowym złożeniem. Dla zwiększenia czytelności kodu zostały one podzielone na poszczególne podsekcje odpowiadające konkretnym obiektom.

2.4.1 Stół

```
//-------------------------------------stół---------------------------------
//noga od stołu
#declare noga=cylinder { <0,0,0>,<0,4.00,0>, 0.30

          scale <1,1,1>
          rotate<0,0,0>
          translate<0,0,0>
}

//blat stołu
#declare blat=box { <-4.60, 4.00, -2.80>,< 4.60, 4.3, 2.80>

      scale<1,1,1>
      rotate<0,0,0>
      translate<0,0,0>
}
#declare wysokosc_stolu=4.3;
```

Rys 18. Kod tworzący elementy stołu

Tworzone są kolejno:

- cylinder wykorzystywany do utworzenia nóg stołu,
- prostopadłościan będący blatem,
- deklarowana jest zmienna wykorzystywana później do łatwiejszego przesuwania obiektów.

2.4.2 Cyfry na kościach

```
//------------------------------cyfry na kościach--------------------------
#declare cyfra1=text {
    ttf "timrom.ttf" "1" 0.1, 0
    scale<0.1,0.1,0.1>
}
#declare cyfra2=text
{
    ttf "timrom.ttf" "2" 0.1, 0
    scale<0.1,0.1,0.1>
}
#declare cyfra3=text
{
    ttf "timrom.ttf" "3" 0.1, 0
    scale<0.1,0.1,0.1>
}
#declare cyfra4=text
{
    ttf "timrom.ttf" "4" 0.1, 0
    scale<0.1,0.1,0.1>
}
#declare cyfra5=text
{
    ttf "timrom.ttf" "5" 0.1, 0
    scale<0.1,0.1,0.1>
}
#declare cyfra6=text
{
    ttf "timrom.ttf" "6." 0.1, 0
    scale<0.1,0.1,0.1>
}
//tablice z numerami na kości wielościenne
#declare num_gorna_polowa_k10 = array[5] {"0", "4", "6", "2", "8"}
#declare num_dolna_polowa_k10 = array[5] {"7", "1", "9", "5", "3"}
#declare num_gorna_polowa_k100 = array[5] {"00", "40", "60", "20", "80"}
#declare num_dolna_polowa_k100 = array[5] {"70", "10", "90", "50", "30"}
```

Rys 19. Kod tworzący cyfry na kościach

Zadeklarowane zostają:

- cyfry od 1-6 z wykorzystaniem polecenia *text*. Parametry podawane do polecenia określają czcionkę tekstu, zawartość, jak również grubość liter, dzięki której możliwe będzie wykonanie „wcięć” w bryle kości z pomocą polecenia *difference*,
- tablice liczb do kości wielościennych dzięki którym możliwe będzie naniesienie napisów na ściany kości z wykorzystaniem pętli *while.* Ułatwia to znacznie złożenie obiektów w późniejszym etapie.

2.4.3 Bryły kości

```
//kość k6 - bryła
#declare kosc_k6= box { <-1.00, 0.00, -1.00>,< 1.00, 2.00, 1.00>

      scale <0.05,0.05,0.05> rotate<0,0,0> translate<0,0,0>
}

//kość k6 z kropkami -bryła
#declare bryla_kostka_6=superellipsoid {
        <0.25, 0.25>
    }

//kość k4 - bryła
#declare  kosc_k4 = object{ Tetrahedron
        scale <1,1,1>*0.045  rotate<0,0,0> translate<0,0,0>
}

//tworzenie połowy kości wielościennej na bazie ostrosłupa
#declare sciany = 5;
#declare wysokosc = .5;
#declare pochylenie = -37;
#declare obrot_scian = 360/sciany;
#declare numer_sciany = sciany;

#declare polowa_kostki_wielosciennej=intersection{
        #while (numer_sciany > 0)
            plane {
                z, 0
                rotate <pochylenie, 0, 0>
                rotate <0,obrot_scian*numer_sciany,0>
                translate <0,wysokosc,0>
            }
            #declare numer_sciany = numer_sciany- 1;
        #end

    }

//kość k10- bryła
#declare kostka_10=intersection{
    object{polowa_kostki_wielosciennej texture {granit_czarny} }
    object{polowa_kostki_wielosciennej scale <1,-1,1> rotate <0,obrot_scian/2,0> texture {granit_czarny} }

}

//kość k100- bryła
#declare kostka_100=intersection{
    object{polowa_kostki_wielosciennej texture {granit_bialy} }
    object{polowa_kostki_wielosciennej scale <1,-1,1> rotate <0,obrot_scian/2,0> texture {granit_bialy} }
```

Rys 20. Kod tworzący bryły kości

Powyższy kod zawiera deklaracje brył poszczególnych kości:

- Kość k6 – utworzona na bazie zwykłego prostopadłościanu, oraz wyskalowana do odpowiednich rozmiarów,
- Kość k6 (wersja z wcięciami zamiast liczb) – utworzona na bazie superelipsoidy, w celu nadania jej zaokrąglonych krawędzi. W zależności od zmiany parametrów wejściowych możliwa jest modyfikacja zaokrąglenia krawędzi,
- Kość k4 – stworzona na bazie czworościanu foremnego. Wyskalowana do odpowiednich rozmiarów,
- Połowa kości wielościennej – stworzenie tej kości wymagało bardzie skomplikowanych operacji. Zadeklarowane na początku zmienne umożliwiają modyfikację parametrów kości (ilości ścian, pochylenia i ich wysokości). Pętla *while* tworzy w kolejnych krokach przesunięte względem siebie płaszczyzny, a następnie wykonuje na nich polecenie *intersection*, które zwraca figurę utworzoną z ich przecięć.
- Kość k10 i k100 – obie te kości tworzone są w identyczny sposób za pomocą złożenia przesuniętych względem siebie oraz obróconych dwóch obiektów *„polowa_kostki_wielosciennej”*. Jedyną zmianą było nadanie im różniących się tekstur.

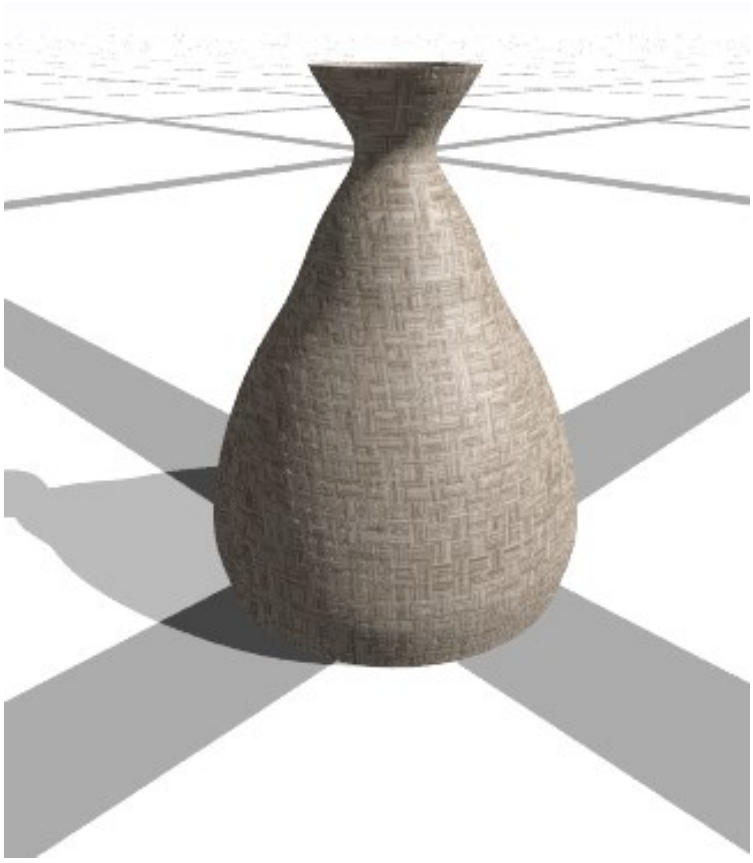
2.4.4 Worek

```
//-------------------------------------worek-------------------------------
//czesc dolna
#declare bag = sor{
  11,
  <0.360,0.000>,   //radius, height
  <0.380,0.120>,
  <0.400,0.240>,
  <0.380,0.360>,
  <0.340,0.480>,
  <0.280,0.600>,
  <0.200,0.720>,
  <0.140,0.840>,
  <0.070,0.960>,
  <0.140,1.080>,
  <0.140,1.100>

    scale 1.0  rotate<0,0,0> translate<0,-0.12,0>
  }

//wyciecie gornej czesci
#declare wycinek = intersection {
   box{ <-2,0,-2.5>,<2.5, 4.00,2.5> }
   object{ Paraboloid_Y }
   //texture{bagTexture}//{ pigment{ color rgb<1, 0.5, 0.7> }
         //  finish { phong 1}
        // }
   scale <1,1,1>*0.5 rotate<0,0,0> translate<0,0.00,0.0>
}
```

Rys 21. Kod tworzący elementy worka

Elementy worka:

- Bag – obiekt powstał przez wykonanie operacji *sor* (obrotu pewnej funkcji względem osi y). Parametry określają ilość punktów do utworzenia obracanej funkcji oraz ich współrzędne.
- Wycinek – utworzony za pomocą wycięcia nakładających się elementów prostopadłościanu oraz paraboloidy symetrycznej względem osi Y.



Rys 22. Woreczek na kości

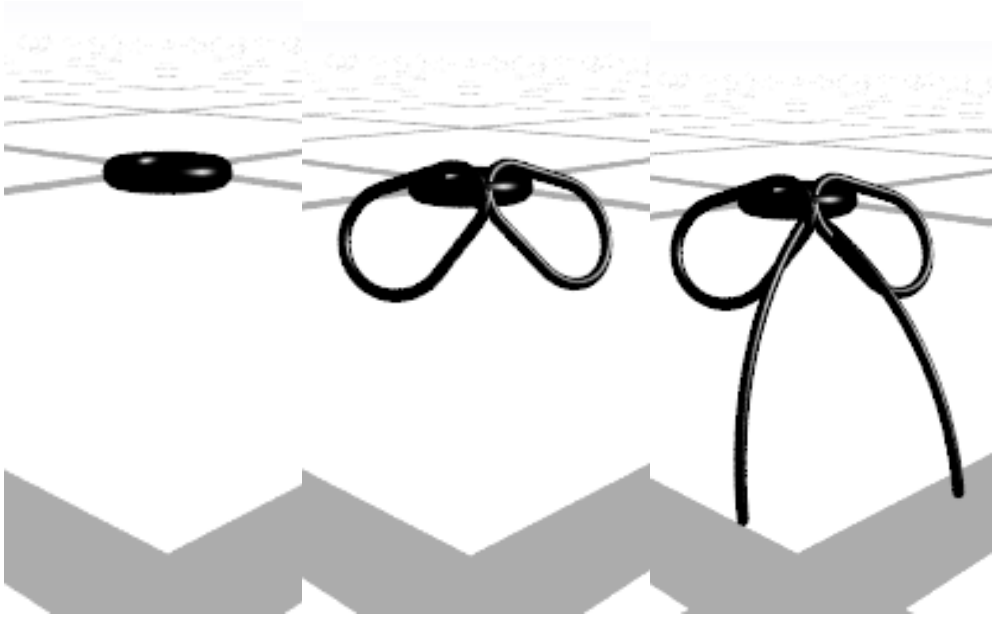
2.4.5 Wiązanie worka

```
//----------------------------------wiazanie---------------------------------
//petla na worku
#declare sznurek = torus { 0.08,0.02
        texture { pigment{ color rgb <0,0.0,0.0>}
                  finish { phong 1 reflection { 0.00 metallic 0.50} }
                } // end of texture
        scale <1,1,1> rotate<0,0,0> translate<0,0.830,0>
      }
//petla kokardy
#declare petla=object{Round_Conic_Torus( 1.00, 0.80, 0.50, 0.10, 0)
         texture{ pigment{ color rgb<0,0,0>}
                  finish { phong 1}
          }
          scale<1,1,1>  rotate<0,0,0>  translate<0,0,0>
        }
//zwis kokardy cz.1
#declare Ball =
 sphere{<0,0,0>,0.01
        scale <1,1,1>
        rotate<0,0,0>
        translate<0,0,0>
            texture{pigment{ color rgb<0,0,0>}
                finish { phong 1}
                    }
        }
//zwis kokardy cz.2
#declare sznurek2=union{
  #declare start=0;
  #declare rozdz=200;
  #declare koniec=2;
        #while(start<rozdz*koniec)
                object{Ball translate <start*0.25/rozdz,0,0>
                rotate <0,0,12/rozdz*start>}
                #local start=start+1;
         #end

}
```

Rys 23. Kod tworzący elementy wiązania

Na wiązanie worka składają się 3 elementy:

- Pętla na worku - utworzona z obiektu *torus* (pierścień o przekroju kolistym),
- Pętla kokardy - obiekt typu *Round Conic Torus* (spłaszczony w kształt stożka pierścień)
- Zwis kokardy – utworzony dwuetapowo poprzez wstępne zadeklarowanie kulki o średnicy odpowiadającej grubości pętli, a następnie, narysowanie z pomocą pętli *while* nakładających się na siebie kolejno kulek, przesuniętych względem siebie i obróconych względem początkowego punktu sznurku, w celu nadania odpowiedniej krzywizny.



Rys 24,25,26. Etapy tworzenia wiązania

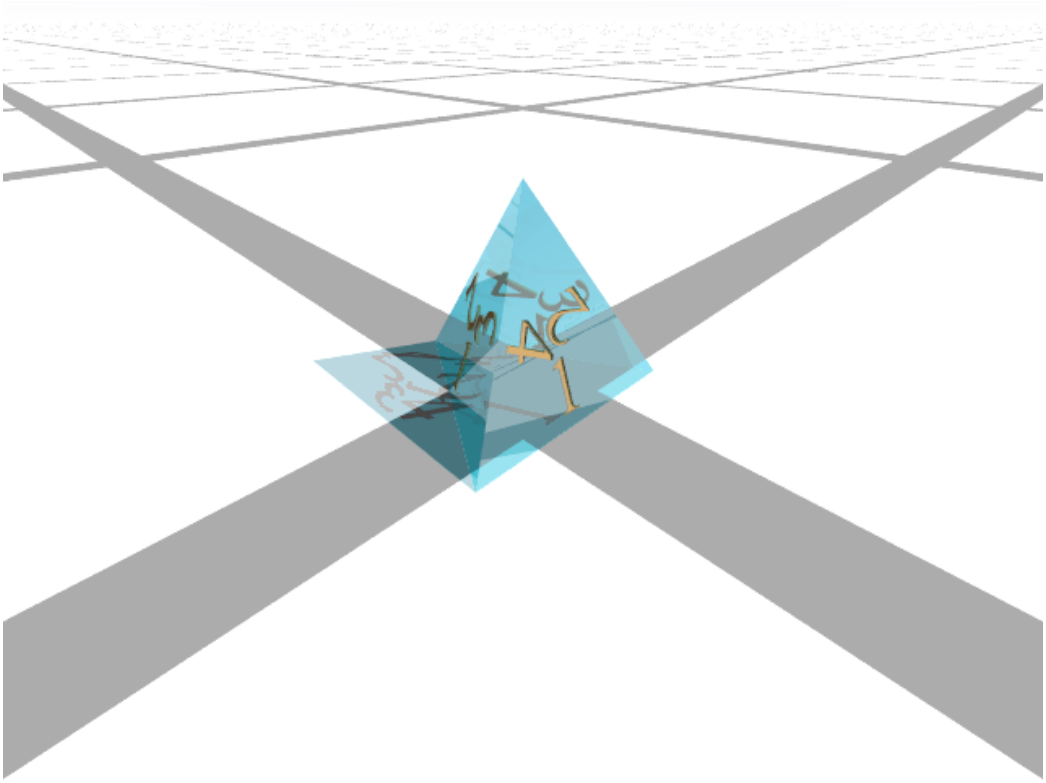
2.5 Sekcja ZŁOŻENIA OBIEKTÓW

2.5.1 Kość k4

```
//---------------------------------kość k4 - całość----------------------------
#declare k4 = union{
difference{
    object {kosc_k4}
    object {cyfra1 pigment { Gold }scale <1,1,1>*0.5 rotate<19.5,0,0>  translate <-0.01,0.01,-0.048>} // #1
    object {cyfra2 pigment { Gold }scale <1,1,1>*0.5 rotate<-8,-19.5,120> translate<0.03,0.06,-0.031>} // #1
    object {cyfra4 pigment { Gold }scale <1,1,1>*0.5 rotate<-8,19.5,-120> translate<-0.03,0.075,-0.025>} // #1


    object {cyfra1 pigment { Gold }scale <1,1,1>*0.5 rotate<19.5,0,0>  translate <-0.01,0.01,-0.048> rotate<0,120,0>} // #2
    object {cyfra3 pigment { Gold }scale <1,1,1>*0.5 rotate<-8,-19.5,120> translate<0.035,0.06,-0.031> rotate<0,120,0>} // #2
    object {cyfra2 pigment { Gold }scale <1,1,1>*0.5 rotate<-8,19.5,-120> translate<-0.03,0.075,-0.025> rotate<0,120,0>} // #2


    object {cyfra1 pigment { Gold }scale <1,1,1>*0.5 rotate<19.5,0,0>  translate <-0.01,0.01,-0.048> rotate<0,-120,0>} // #3
    object {cyfra4 pigment { Gold }scale <1,1,1>*0.5 rotate<-8,-19.5,120> translate<0.037,0.06,-0.031> rotate<0,-120,0>} // #3
    object {cyfra3 pigment { Gold }scale <1,1,1>*0.5 rotate<-8,19.5,-120> translate<-0.03,0.075,-0.025> rotate<0,-120,0>} // #3


    object {cyfra3 pigment { Gold }scale <1,1,1>*0.5 rotate<-90,180,0>  translate <0.01,-0.048,-0.043>} // #4
    object {cyfra2 pigment { Gold }scale <1,1,1>*0.5 rotate<-90,60,0> translate<0.035,-0.048,0.03>} // #4
    object {cyfra4 pigment { Gold }scale <1,1,1>*0.5 rotate<-90,-60,0> translate<-0.045,-0.048,0.01>} // #4
    }
}
```

Rys 27. Połączenie kości k4

Złożenie kości k4 polegało na połączeniu bryły kości z zestawem cyfr, po trzy na każdej ścianie w odpowiedniej konfiguracji i odpowiednim przesunięciu/rotacji. Efekt złożenia:



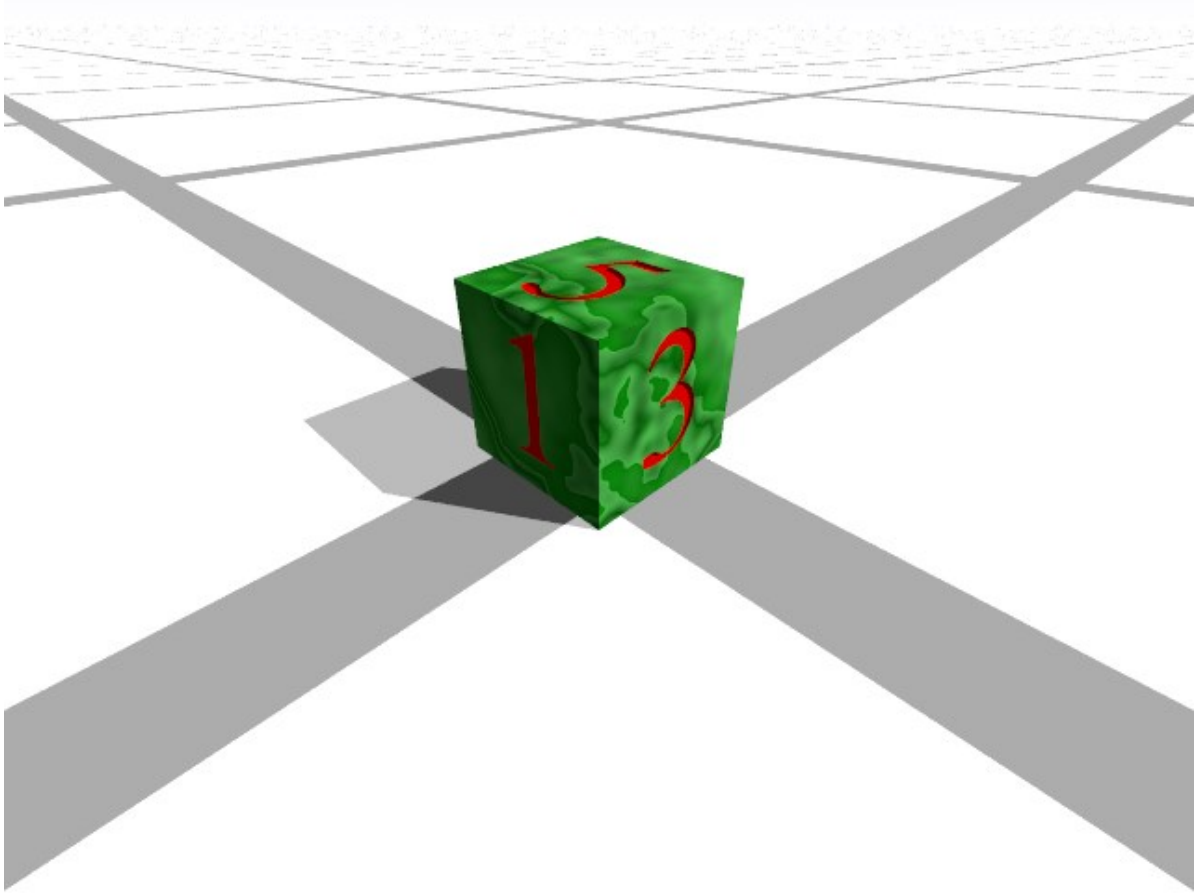
Rys 28. Kość k4 – złożenie

2.5.2 Kość K6 z cyframi

```
//------------------------------kość k6 - całość--------------------------------
#declare k6 = union{
difference{
    object {kosc_k6}
    object {cyfra1 pigment {Red }rotate<0,90,0>  translate <-0.058,0.02,0.01>}
    object {cyfra2 pigment {Red }rotate<0,-90,-90> translate<-0.035,-0.008,-0.03>}
    object {cyfra3 pigment {Red }translate <-0.02,0.02,-0.058>}
    object {cyfra4 pigment {Red }rotate<180,0,-90> translate<0.035,0.078,0.058>}
    object {cyfra5 pigment {Red }rotate<0,90,-90> translate<-0.035,0.108,0.02>}
    object {cyfra6 pigment {Red }rotate<0,-90,0> translate<0.058,00.02,-0.02>}
    }
}
```

Rys 29. Połączenie kości k6 z cyframi

Złożenie kości k6 polegało na połączeniu bryły kości z zestawem sześciu cyfr (po jednej na ścianę) po ich uprzednim obrocie. Efekt złożenia:



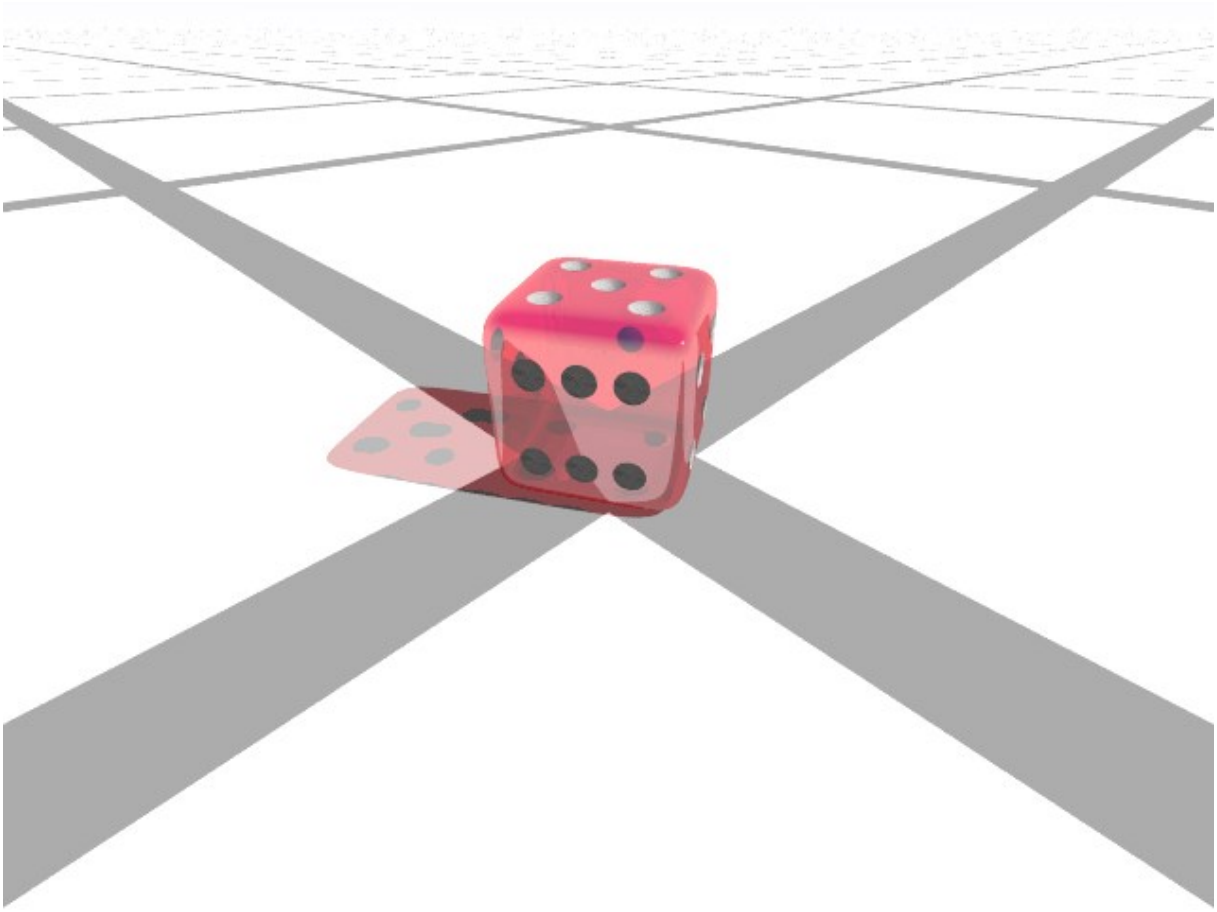
Rys 30. Kość k6 z cyframi – złożenie

2.5.3 Kość K6 z kropkami

```
//-----------------------kość k6_z kropkami - całość-------------------------
#declare kostka_6=union{
 difference {
    object { bryla_kostka_6}
      //1
      object { kropka translate <0,0,-zewn> }
      //2
      object { kropka translate <-na_scianie,-zewn,-na_scianie> }
      object { kropka translate <na_scianie,-zewn,na_scianie> }
      //3
      object { kropka translate <zewn,-na_scianie,-na_scianie> }
      object { kropka translate <zewn,na_scianie,na_scianie> }
      object { kropka translate <zewn,0,0> }
      //4
      object { kropka translate <-zewn,-na_scianie,-na_scianie> }
      object { kropka translate <-zewn,-na_scianie,na_scianie> }
      object { kropka translate <-zewn,na_scianie,na_scianie> }
      object { kropka translate <-zewn,na_scianie,-na_scianie> }
      //5
      object { kropka translate <0,zewn,0> }
      object { kropka translate <-na_scianie,zewn,-na_scianie> }
      object { kropka translate <-na_scianie,zewn,na_scianie> }
      object { kropka translate <na_scianie,zewn,-na_scianie> }
      object { kropka translate <na_scianie,zewn,na_scianie> }

      //6
      object { kropka translate <0,-na_scianie,zewn> }
      object { kropka translate <0,na_scianie,zewn> }
      object { kropka translate <-na_scianie,-na_scianie,zewn> }
      object { kropka translate <na_scianie,-na_scianie,zewn> }
      object { kropka translate <-na_scianie,na_scianie,zewn> }
      object { kropka translate <na_scianie,na_scianie,zewn> }

        scale <1,1,1>*0.05
        translate <0,0.05,0>
        }
}
```

Rys 31. Połączenie kości k6 z kropkami

Złożenie kości k6 polegało na połączeniu bryły kości z zestawem sześciu cyfr (po jednej na ścianę) po ich uprzednim obrocie. Efekt złożenia:



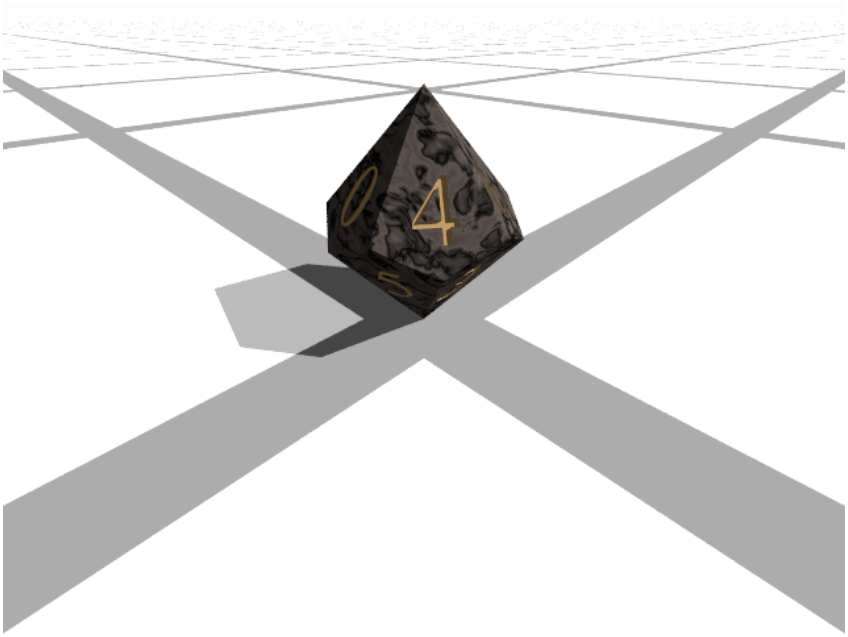
Rys 32. Kość k6 z kropkami – złożenie

2.5.4 Kość K10

```
//--------------------------------kosc k10 - całość---------------------------------
#declare k10 = union{
 difference {
object {kostka_10 rotate<0,0,0> translate <0,0,0>}
//bryła
#declare numer_sciany = sciany;

    #while (numer_sciany > 0)

        #declare numer_gorny_k10 = num_gorna_polowa_k10[numer_sciany-1]
        #declare numer_dolny_k10 = num_dolna_polowa_k10[numer_sciany-1]
            //gorne numery
            text {
                ttf "timrom",
                numer_gorny_k10
                wysokosc/10, 0
                translate <-.25,0,0>
                scale .35
                scale <-1,1,1>
                translate <0, 0, wysokosc/1.6>
                rotate <pochylenie,0,0>
                translate <0, -wysokosc*.3,.08>
                rotate <0,-obrot_scian*numer_sciany,0>
                    pigment {
                        color Gold
                    }
                }
            //dolne nummery
            text {
                ttf "timrom"
                numer_dolny_k10
                wysokosc/10, 0
                translate <-.25,0,0>
                scale .35
                scale <-1,1,1>
                translate <0,0, wysokosc/1.6>
                rotate <-pochylenie,0,0>
                translate <0, -wysokosc*.18, -.1>
                rotate <0, 180-obrot_scian*numer_sciany,0>
                    pigment {
                        color Gold
                    }
                }

        #declare numer_sciany = numer_sciany - 1;
    #end
    }
  scale <1,1,1>*0.17
  translate <0,0.085,0>
```

Rys 33. Połączenie kości k10

Utworzenie kości polegało na połączeniu bryły kości k10 oraz numerów, lecz w tym wypadku wykorzystana została do tego pętla while. W kolejnych iteracjach pętli zmieniany jest element tablicy z którego pobierany jest tekst do nałożenia na ścianę kości, po czym następuje obrót kości do następnej iteracji. Operacja jest wykonywana na obu połówkach kości. Efekt końcowy:



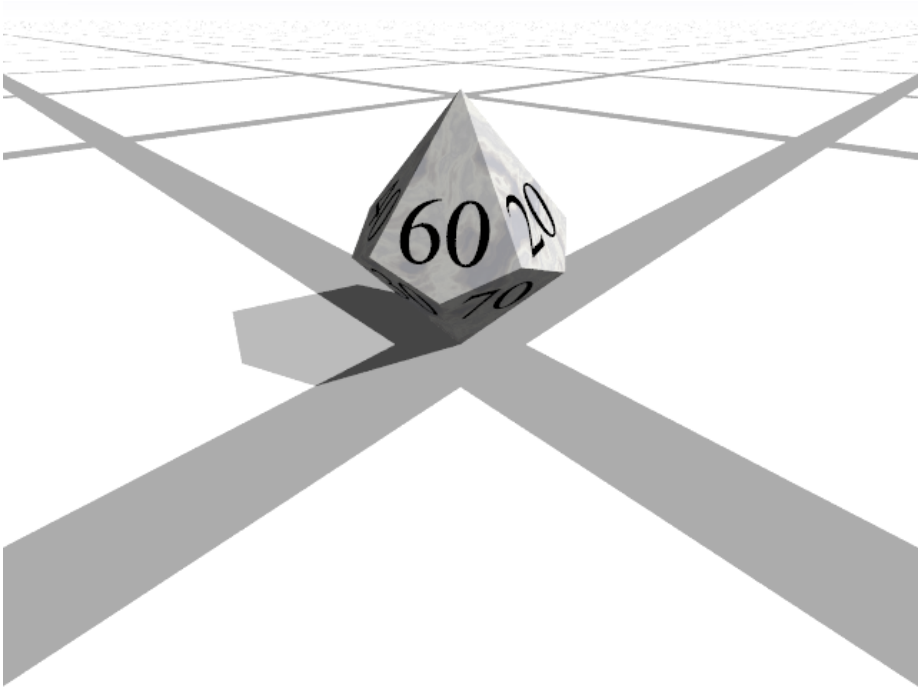
Rys 34. Kość k10 – złożenie

2.5.5 Kość K100

```
//--------------------------------kosc k100 - całość---------------------------------
#declare k100 = union{

//bryła
object {kostka_100 rotate<0,0,0> translate <0,0,0>}

#declare numer_sciany = sciany;

    #while (numer_sciany > 0)

        #declare numer_gorny_k100 = num_gorna_polowa_k100[numer_sciany-1]
        #declare numer_dolny_k100 = num_dolna_polowa_k100[numer_sciany-1]
            //gorne numery
                text {
                    ttf "timrom",
                    numer_gorny_k100
                    wysokosc/10, 0
                    translate <-.5,0,0>
                    scale .30
                    scale <-1,1,1>
                    translate <0, 0, wysokosc/1.6>
                    rotate <pochylenie,0,0>
                    translate <0, -wysokosc*.3,.08>
                    rotate <0,-obrot_scian*numer_sciany,0>
                        pigment {
                            color Black
                        }
                    }
            //dolne nummery
                text {
                    ttf "timrom"
                    numer_dolny_k100
                    wysokosc/10, 0
                    translate <-.5,0.3,0>
                    scale .3
                    scale <-1,1,1>
                    translate <0,0, wysokosc/1.6>
                    rotate <-pochylenie,0,0>
                    translate <0, -wysokosc*.18, -.1>
                    rotate <0, 180-obrot_scian*numer_sciany,0>
                        pigment {
                            color Black
                        }
                    }

        #declare numer_sciany = numer_sciany - 1;
    #end
 scale <1,1,1>*0.17
  translate <0,0.085,0>
```

Rys 35. Połączenie kości k100

Tworzenie kości k100 jest analogiczne do tworzenia kości k10 z różnicą w tablicy liczb do wpisania na ścianę kości oraz przesunięć napisów na ścianie kości, aby umożliwić poprawne wyświetlenie liczb dwucyfrowych. Efekt:



Rys 36. Kość k100 – złożenie

2.5.6 Sakiewka

```
//------------------------------------------------------------------------------
//-------------------------------ZŁOŻENIE WORKA---------------------------------
//------------------------------------------------------------------------------

//---------------------------------worek - całość--------------------------------
#declare worek=difference {
object {bag texture {fabric} rotate<0,0,0> translate <0,0,0>}
object {wycinek texture {fabric} scale<1,1,1>*0.18 translate<0,0.8,0>}
}
//---------------------------------sznurek - całość-------------------------------
#declare wiazanie=union{
object {sznurek rotate<0,0,0> translate<0,0,0>}
object {petla  scale<1,1,1>*0.1 rotate<0,-36,120>translate<-0.05,0.82,-0.08>}
object {petla  scale<1,1,1>*0.1  rotate<0,36,-120>translate<0.05,0.82,-0.08>}
object{sznurek2 rotate<0,36,-144> translate<0,0.82,-0.08>}
object{sznurek2 rotate<180,36,180+144> translate<00,0.82,-0.08>}
}

#declare sakiewka=union{
object{wiazanie}
object{worek}
}
```

Rys 37. Połączenie sakiewki

Sakiewka powstaje poprzez połączenie dwóch obiektów:

- Worek – powstałego z różnicy zbiorów wykonanej na wcześniej opisanych obiektach bag i wycinek oraz nałożeniu na nie tekstury fabric,
- Wiązanie – składającego się z opisanych wcześniej elementów: sznurek, dwóch pętli w odbiciu lustrzanym względem siebie, oraz dwóch zwisów sznurka , również odbitych lustrzanie.

Efekt złożenia:

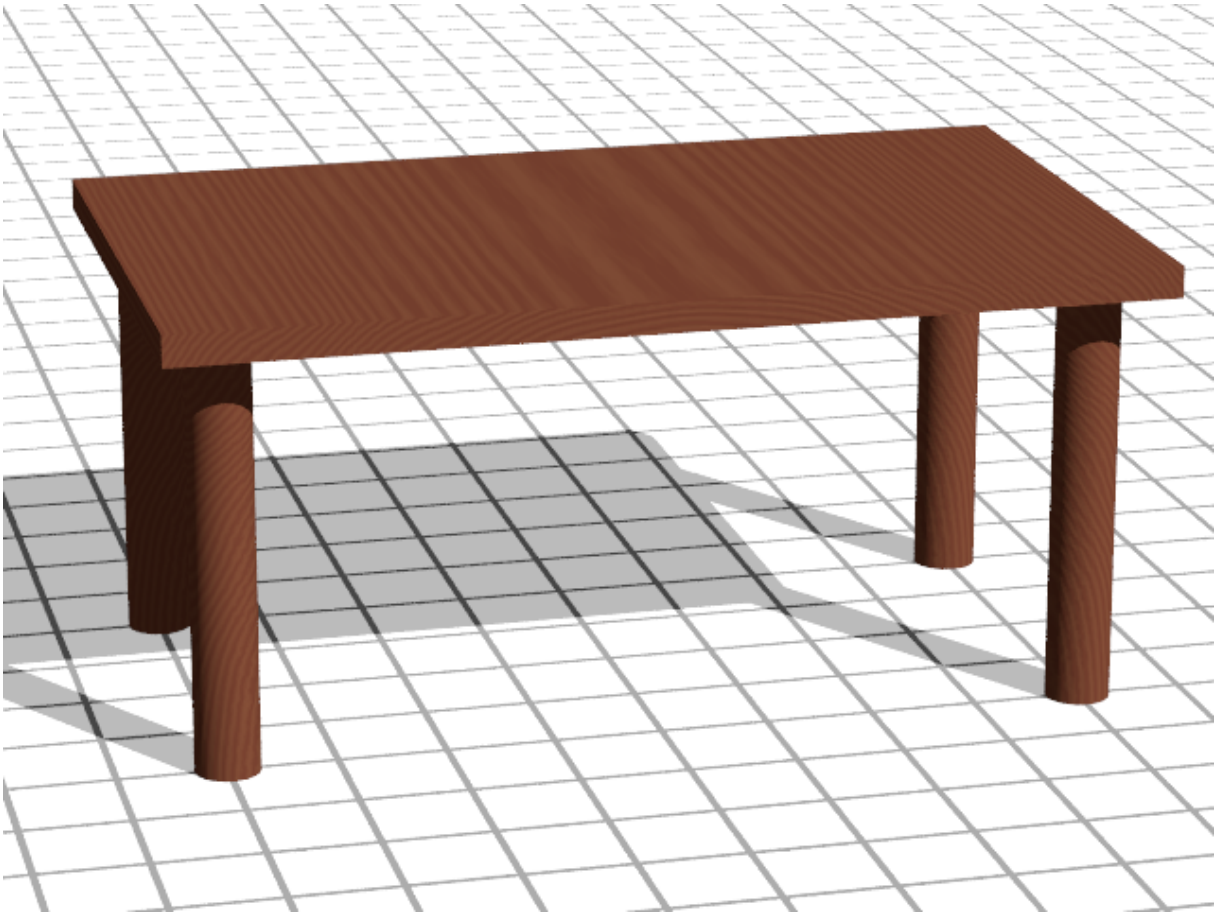
Rys 38. Woreczek na kości ze sznurkiem – złożenie

2.5.7 Stół

```
//------------------------------------------------------------------------------
//-------------------------------ZŁOŻENIE STOŁU---------------------------------
//------------------------------------------------------------------------------

#declare stol = union{
object {noga translate <-4,0,-2>}
object {noga translate <4,0,-2>}
object {noga translate <4,0,2>}
object {noga translate <-4,0,2>}
object {blat}
texture{drewno3}
}
```

Rys 39. Połączenie stołu

Stół został złożony poprzez połączenie czterech obiektów noga przesuniętych względem siebie na płaszczyźnie XZ, obiektu blat oraz nałożenie tekstury drewna. Efekt złożenia:



Rys 40. Stół - złożenie

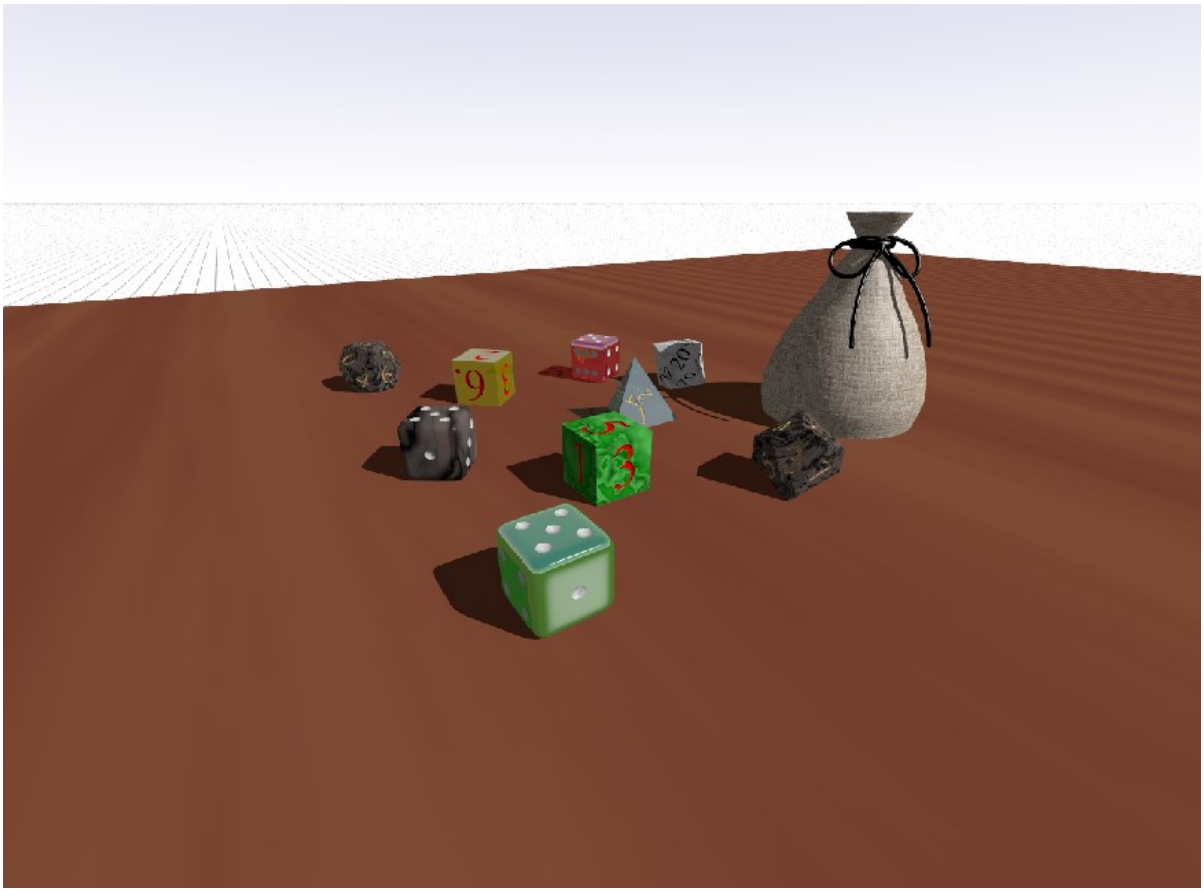
3. Ostateczny wygląd

Ostatnim etapem projektu było połączenie wszystkich stworzonych elementów w całość, co zostało wykonane w poniższym kodzie:

```
//------------------------------------------------------------------------
//--------------------------- obiekty na scenie --------------------------
//------------------------------------------------------------------------

object {stol}
union{

object {k6 texture {granit_zielony} rotate<0,0,0> translate <0, wysokosc_stolu, 0>}
object {k6 texture {szklo_zolte} rotate<0,45,180> translate <0, wysokosc_stolu+0.1, 0.5>}
object {k4 texture {szklo_niebieskie} rotate <0,0,0> translate<0.2,wysokosc_stolu,0.2>}
object {kostka_6 texture {granit_czarny} rotate<0,35,0> translate <-0.2,wysokosc_stolu,0.2>}
object {kostka_6 texture {szklo_zielone} rotate<0,0,0> translate <-0.2,wysokosc_stolu,-0.2> }
object {kostka_6 texture {szklo_czerwone} rotate<0,240,0> translate <0.3,wysokosc_stolu,0.5> }
object {k100 rotate<-(90+pochylenie),120,0> translate <0.5,wysokosc_stolu,0.3>}
object {k10 rotate<-(90+pochylenie),0,0> translate <-.2,wysokosc_stolu,0.8>}
object {k10 rotate<(90+pochylenie),55,0> translate <0.2,wysokosc_stolu,-0.2>}
object {sakiewka scale <1,1,1>*0.4 rotate<0,30,0> translate <0.6,wysokosc_stolu,0>}
}
```

Rys 41. Utworzenie końcowej sceny z obiektami

Możemy zauważyć że konieczne było przesunięcie kości względem siebie, a także ich obrócenie aby były skierowane w różne strony. Translacja w osi Y wykorzystuje wcześniej zadeklarowaną zmienną wysokosc_stolu. W przypadku kości k10 i k100 było również konieczne ich pochylenie względem osi Y, aby położyły się na stole zamiast spoczywać na wierzchołku. Widzimy również, że poprzez predefiniowanie tekstur możliwe było utworzenie dwóch obiektów tej samej klasy z różnymi teksturami. Scena końcowa prezentuje się w następujący sposób:



Rys 42. Scena przedstawiająca kości oraz woreczek na stole

Scena po dodaniu bryły pokoju oraz tekstur ścian została umieszczona na osobnej (ostatniej) stronie dokumentu.