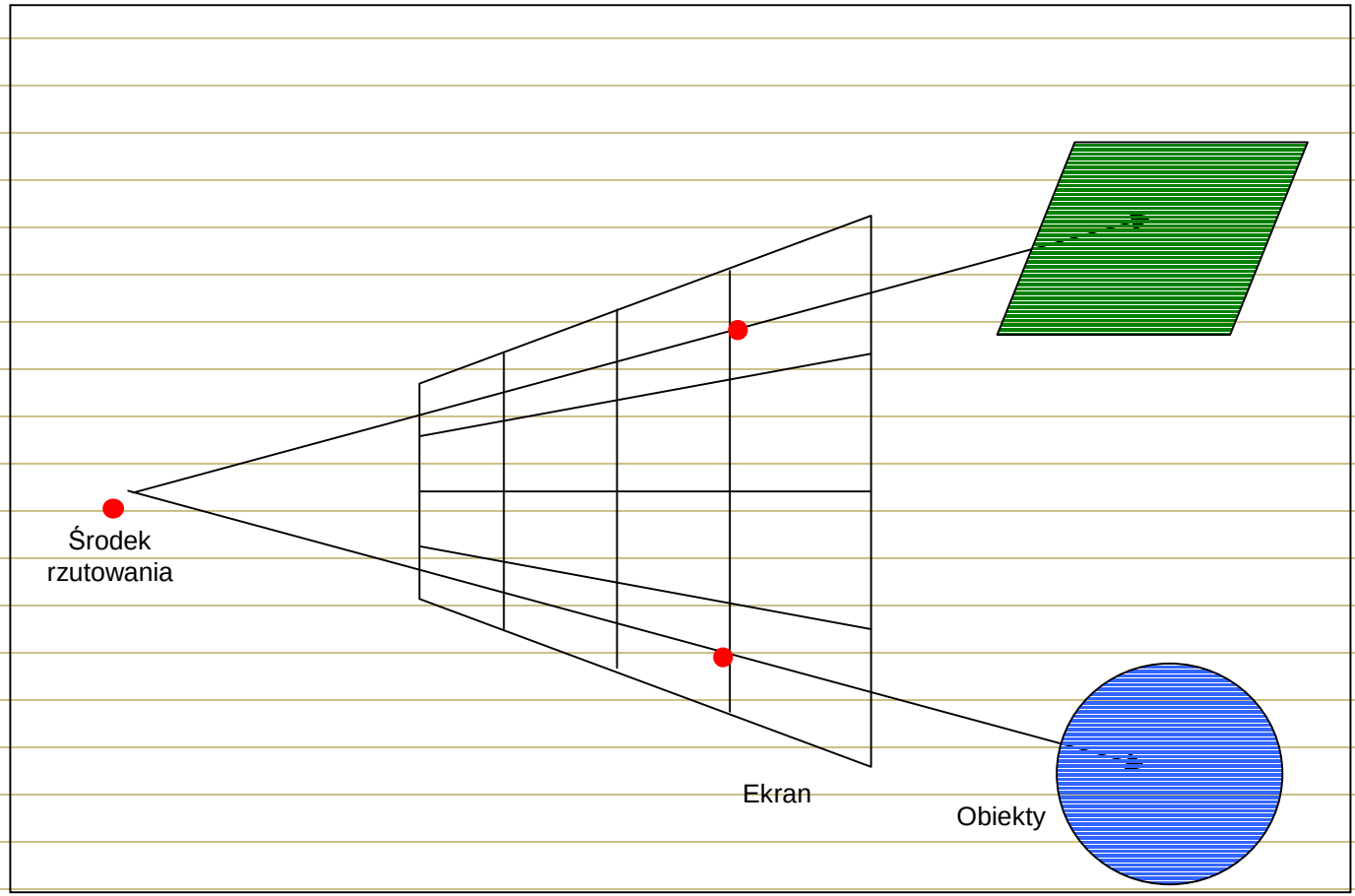
Grafika Komputerowa

POV-RAY

Co to jest metoda Ray-Tracing

- Ray-Tracing jest techniką modelowania scen w grafice trójwymiarowej (grafika 3D) poprzez symulację drogi promieni świetlnych emitowanych od źródła światła lub jasnych (oświetlonych) obiektów do innych obiektów w scenie.
- Ray-Tracing w dosłownym tłumaczeniu oznacza „*śledzenie promieni*”. Technikę Ray-Tracingu można inaczej określić jako „*programowe obliczanie rozkładu cieni i refleksów światła z różnych źródeł*

Graficzne przedstawienie metody Ray Tracing



Pseudokod dla prostej metody śledzenia promieni

```
wybranie środka rzutowania i pola wizualizacji na rzutni
for (każdy przeglądany wiersz obrazu)
{
    for (każdy piksel w przeg1ądanym wierszu)
    {
        wyznaczenie promienia ze środka rzutowania przez piksel;
        for (każdy obiekt w scenie)
        {
            if (obiekt jest przecinany i jest najbliższy z dotychczas
            rozważanych rejestrowanie przecięcia i nazwy obiektu;
        }
        ustawienie barwy piksela zgodnie z barwą najbliższego przeciętego
        obiektu;
    }
}
```

Jakie problemy występują w metodzie Ray-Tracing? Problemy geometryczne

- Obliczanie przecięć: np. kula z promieniem
- Równanie kuli: $(x-x0)^2+ (y-y0)^2+ (z-z0)^2=r^2$
- Równanie promienia: $P(t)=P_0+t(P_1-P_0)$
- Trzeba to równanie rozwiązać i wyznaczyć punkty przecięcia.
- Podobnie postępujemy z innymi bryłami.

Jakie problemy występują w metodzie Ray-Tracing ? - Model oświetlenia

- I = ambient + diffuse * color * Σ (color_Lj * N°Lj) + phong * Σ(N°Lj)$^{phong\_size}$ + reflection * Ir + refraction * If
- Gdzie:
  - *ambient* - symulacja światła "tła",
  - *diffuse* - współczynnik dyfuzyjnego odbicia światła,
  - *color* - kolor oświetlanej powierzchni,
  - *N* - wektor normalny do oświetlanej powierzchni,
  - *color_Lj* - kolor wysyłany przez j-te źródło światła,
  - *Lj* - wektor do j-tego źródła światła,
  - *phong* - współczynnik odblasku Phonga,
  - *phong_size* - wykładnik odblasku Phonga,
  - *Lj'* - wektor idealnego odbicia j-tego źródła światła od oświetlanej powierzchni,
  - *reflection* - współczynnik odbicia lustrzanego,
  - *Ir* - ilość i kolor światła dochodzącego z kierunku odbicia lustrzanego,
  - *refraction* - współczynnik przezroczystości,
  - *If* - ilość i kolor światła dochodzącego z kierunku załamania,

Interpretacja współczynników w metodzie Ray-Tracing:

- *ambient* - gdy jest równy 0, wtedy cienie (nieoświetlone fragmenty przestrzeni, przesłonięte przez obiekty) będą idealnie czarne; gdy jest różny od zera, wtedy powierzchnia w cieniu będzie delikatnie widoczna; wartości dopuszczalne: <0,1>,
- *diffuse* - określa, ile procent padającego światła ulega rozproszeniu; powierzchnie matowe rozpraszają prawie 100% padającego światła; wartośi dopuszczalne: <0,1>,
- *phong, phong size* - definiuje połysk (gładkość) powierzchni, jeżeli parametr "phong" posiada wartość większą od 0, wtedy na powierzchni powstają odblaski będące wynikiem lustrzanego odbicia źródła światła; parametr "phong" określa, ile procent światła padajęcego na powierzchnię uczestniczy w odblasku (zwykle 90-100%), parametr "phong_size" określa rozmiar elipsy odblasku - im większa wartość "phong_size", tym mniejsza elipsa odblasku; wartości dopuszczalne dla "phong": <0,1>, dla "phong size": <5,240>,
- *reflection* - również definiuje połysk powierzchni - określa, czy powierzchnia ma charakter lustrzany; wartość 1 oznacza idealne lustro - odbijające 100% padającego od innych obiektów światła; wartości dopuszczalne: <0,1>,
- *refraction* - definiuje przezroczystość powierzchni - określa, jak wiele światła padającego od innych obiektów przedostaje się przez powierzchnię; wartość 1 oznacza doskonałą przezroczystość; wartości dopuszczalne: <0,1>; najczęściej wraz z parametrem "refraction" definiuje się współczynnik załamania światła "ior" (ang. index of refraction), który wprost wpływa na kąty załamania światła przechodzącego przez powierzchnię, a jego wartość zależy od fizycznego materiału (np. szkło ołowiowe posiada ior=1.51).

Implementacja

- Program Pov-Ray

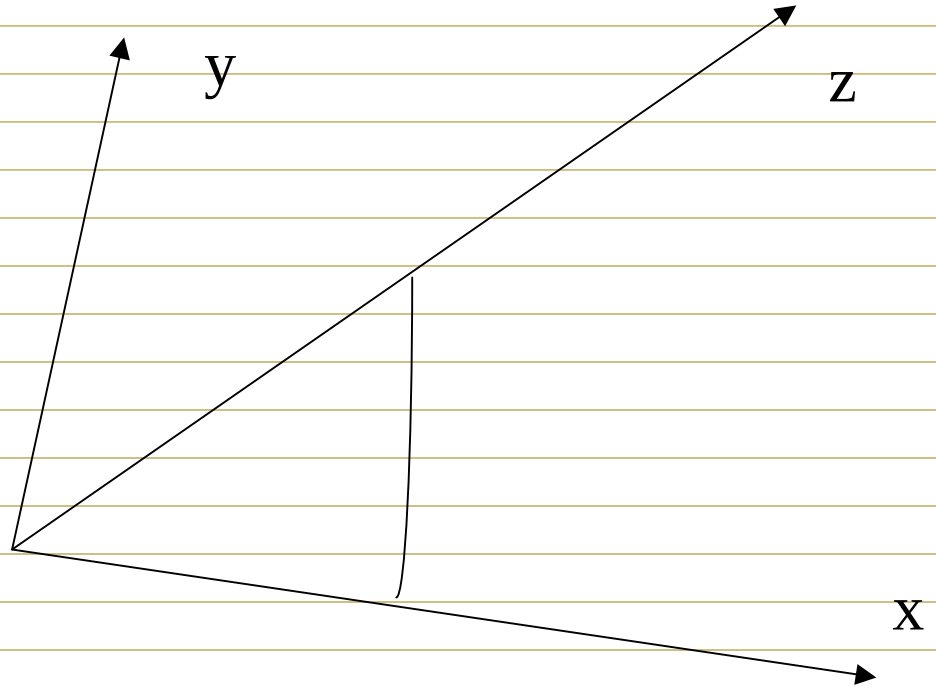
Do czego służy Pov-Ray

- Używa metody Ray-tracingu
- Łatwy język opisu sceny
- dużo przykładowych scen
- zawiera wiele plików z predefiniowanymi krzywiznami, kolorami, teksturami
- końcowe rysunki mogą być bardzo dobrej
- wiele różnych typów światła (liniowe, punktowe, powierzchniowe), perspektyw, tekstur,

Do czego służy Pov-Ray

- wiele różnych efektów atmosferycznych: tęcza, mgła,
- różne modele rzeczywistych obiektów: ogień, chmury,
- możliwość generowania rysunku w różnych formatach,
- podstawowe obiekty: kule, prostopadłościany, stożki, powierzchnie,
- zaawansowane obiekty: tekst, krzywe Beziera, „bloby”,

Układ współrzędnych



Dołączanie plików

- Dyrektywa include
- Są to pliki tekstowe
- Przykłady
  - #include „colors.inc” - zdefiniowane kolory
  - #include "textures.inc” - tekstury
  - #include "shapes.inc” - krzywizny
  - #include "glass.inc” - rodzaje szkła
  - #include "metals.inc” - rodzaje metali
  - #include "woods.inc” - rodzaje drewna

Przykład pliku dołączanego

```
declare Red     = rgb <1, 0, 0>;
#declare Green   = rgb <0, 1, 0>;
#declare Blue    = rgb <0, 0, 1>;
#declare Yellow  = rgb <1,1,0>;
#declare Cyan    = rgb <0, 1, 1>;
#declare Magenta = rgb <1, 0, 1>;
#declare Clear   = rgbf 1;
#declare White   = rgb 1;
#declare Black   = rgb 0;
#declare Gray05 = White*0.05;
#declare Gray10 = White*0.10;
#declare Gray15 = White*0.15;
```

Schemat tworzenia sceny

```
#include ”colors.inc”
light_source{
  <3,2,-5>
  color rgb <1,1,1>
}  // źródło światła
camera{
  location <0,0,-3>
  look_at 0
}  //położenie kamery
sphere{
0 1
texture{pigment{color rgb <1,0.2,0.2>}}
}  //obiekt
```

Kamera

camera{ [CAMERA_ITEMS...] }

CAMERA_ITEM:
CAMERA_TYPE | CAMERA_VECTOR

CAMERA_TYPE:
perspective | ultra_wide_angle | panoramic | cylinder

CAMERA_VECTOR:
location <Location> | right <Right> | up <Up> | direction <Direction> |
sky <Sky>

Postać kamery

- Postać ogólna:
- **camera**
- **{**
- **location <x, y, z>**
- **look_at <x', y', z'>**
- **}**
- gdzie:
- <x, y, z> – wektor położenia kamery (obserwatora),
- <x', y', z'> – wektor położenia punktu, na który kamera jest zwrócona.

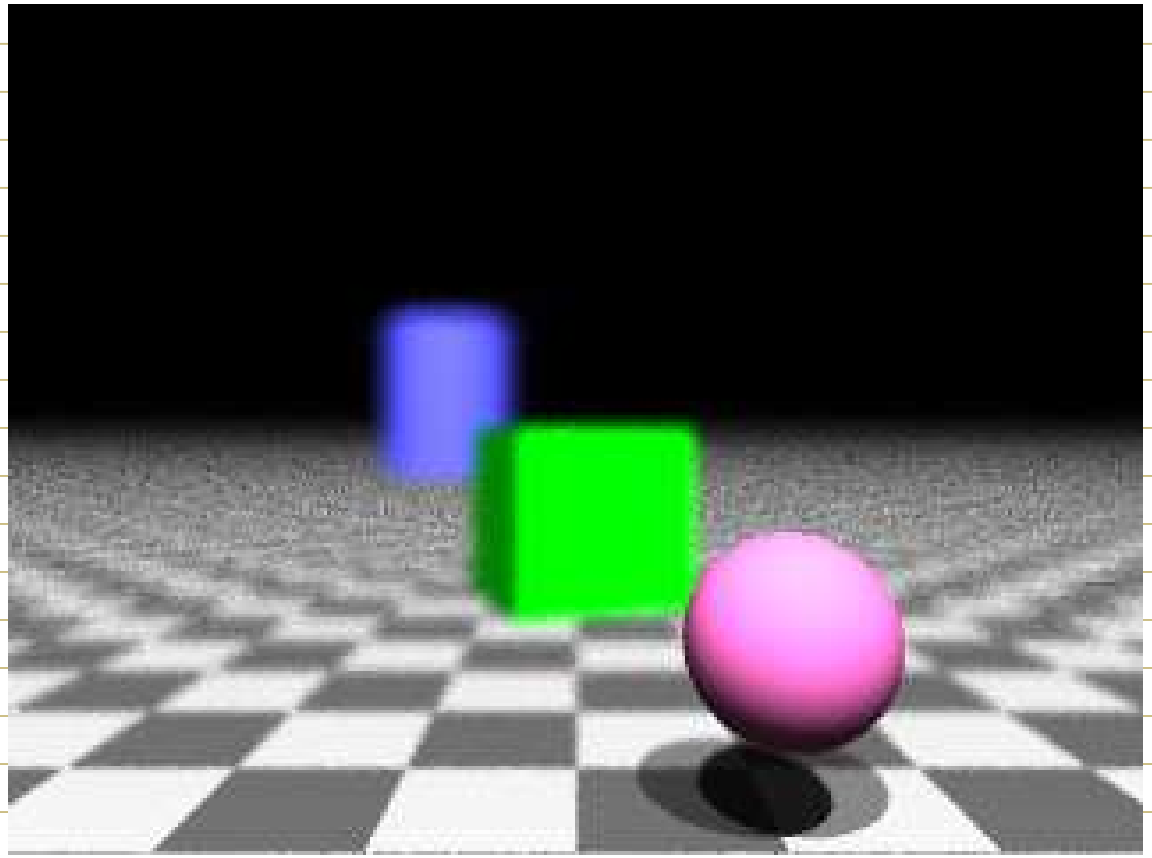
Właściwości kamery

- Efekt głębi ostrości najłatwiej można zauważyć oglądając fotografie. Widzimy na nich główne obiekty jako ostre (np. postacie) i elementy otoczenia, które często są rozmyte, (np. tło, plener). Oczywiście głębia ostrości, jej zasięg i przedział jest uzależniona od ustawień obiektywu aparatu.
- Kamera programu POV-Ray, w celu tworzenia realistycznie wyglądających scen, również została wyposażona w ustawienia (tzn. parametry) dotyczące głębi ostrości. Odpowiadają za to słowa kluczowe focal_point, aperture, blur_samples.
  - Parametr focal_point wyposażony w wektor <x, y, z> określa punkt, w którym rozpoczyna się głębia ostrości. W powyższym przykładzie parametr ustawiony jest na różową kulę, która jest ostra.
  - Parametr aperture określa przedział (zakres) głębi ostrości. Wartość tego atrybutu ustawiona na 0.05 oznacza bardzo mały przedział głębi co w efekcie doprowadza do rozmycia wszystkich obiektów w scenie, zaś wartość 1.5 definiuje bardzo dużą głębię i ostrość wszystkich obiektów.
  - Parametr blur_samples określa jakość generowanego obrazu – czym wyższa wartość (np. blur_samples 50) tym wyższa jakość obrazu wynikowego ale i dłuższy czas renderingu.

Przykład

```
#include "colors.inc"
#include "shapes.inc"
#include "textures.inc"
sphere
{     <1, 0, -6>, 0.5
      finish
      {
            ambient 0.1
            diffuse 0.6
      }
      pigment { NeonPink }
}
box
{
      <-1, -1, -1>, < 1, 1, 1>
      rotate <0, -20, 0>
      finish
      {
            ambient 0.1
            diffuse 0.6
      }
      pigment { Green }
}
cylinder
{
      <-6, 6, 30>, <-6, -1, 30>, 3
      finish
      {
            ambient 0.1
            diffuse 0.6
      }
      pigment {NeonBlue}
}
plane
{
      y, -1.0
      pigment
       {
            checker color Gray65 color Gray30
      }
}
light_source { <5, 30, -30> color White }
light_source { <-5, 30, -30> color White }
camera
{
      location <0.0, 1.0, -10.0>  look_at <0.0, 1.0, 0.0>
      focal_point < 1, 1, -6> aperture 0.5
      blur_samples 50
}
```

Wynik przykładu



Definiowanie źródła światła

- Ważnym elementem niezbędnym do wymodelowania sceny jest światło. Pozwala ono dostrzec obiekty znajdujące się w scenie oraz ich szczegóły. Każda scena, nie posiadająca źródła światła zostanie wygenerowana jako czarny obraz. Źródło światła to nieskończenie mały punkt, umieszczony w określonym punkcie trójwymiarowego układu współrzędnych emitujący światło o określonej barwie.

Źródło światła

light_source { <Location>, COLOR [LIGHT_MODIFIERS...] }

LIGHT_MODIFIER:
LIGHT_TYPE | AREA_LIGHT_ITEMS

LIGHT_TYPE:
spotlight | shadowless | cylinder

AREA_LIGHT_ITEM:
area_light <Axis_1>, <Axis_2>, Size_1, Size_2 |

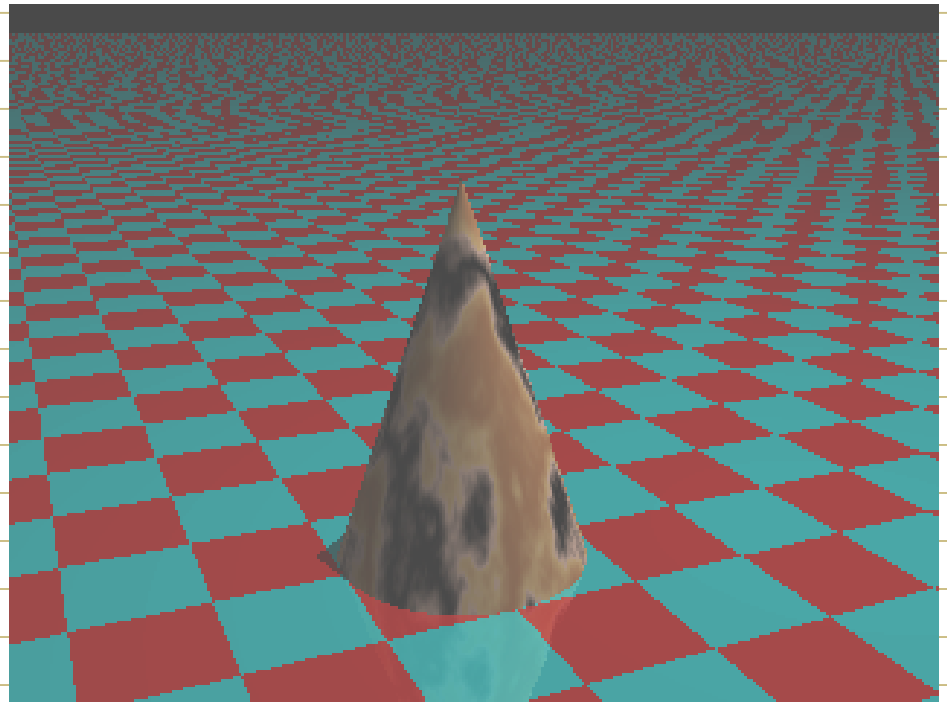
Definicja źródła światła

- Ogólna postać definicji źródła światła:
- **light_source { <x, y, z> color rgb < r,g,b > }**
- gdzie:
- x, y, z – składowe wektora położenia źródła światła,
- r, g, b – składowe wektora określające kolor światła.
- Punkt świetlny (ang. pointlight) jest punktem emitującym światło. Punkt ten jest niewidoczny i oświetla jednakowo wszystkie obiekty w scenie, niezależnie od ich oddalenia (zachowania te można jednak zmienić). Punkt świetlny jest podstawowym źródłem światła i składa się z dwóch parametrów: location (położenie) i color (barwa).

Przykład

```
#include "colors.inc"
#include "textures.inc"
camera
{
	Location <-4, 3, -9> look_at <0, 0, 0>		angle 48
}
	plane
	{
		y, -1 texture { pigment { checker  color rgb<0.5, 0, 0>  color rgb<0, 0.5, 0.5>}
		finish
		{
			diffuse 0.4   ambient 0.2  phong 1  phong_size 100  reflection 0.25
		}
	}
}
cone
{
	<0,1,0>, 0, <0,0,0>, 1
	texture { Brown_Agate }
	scale <1, 3, 1>
	translate <-1, -1, -2>
}
light_source { <2, 10, -3> color White }
```

Wynik renderingu



Źródła światła - Reflektory

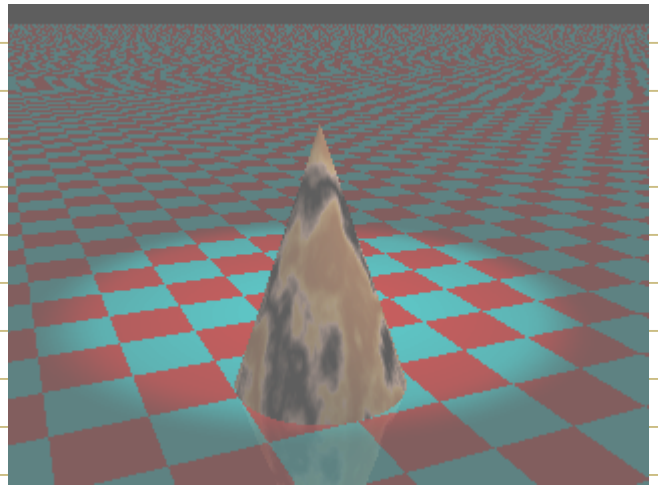
- **Reflektory (spotlight)**
- Postać ogólna:
- **light_source**
- **{**
- **<x, y, z> color rgb <r, g, b>**
- **spotlight**
- **radius m**
- **falloff n**
- **tightness p**
- **point_off <x', y', z'>**
- **}**
- gdzie:
- x, y, z – składowe wektora położenia źródła światła,
- r, g, b – składowe wektora koloru źródła światła,
- m – kąt nachylenia krawędzi bocznej stożka światła,
- n – kąt stożka w miejscu rozpadu (wygaszenia) promieni światła,
- p – wysokość, na której następuje rozpad światła,
- x', y', z' – składowe wektora punktu padania światła.

Reflektory

- Reflektory (ang. spotlight) są bardzo użytecznym typem źródła światła. Służą do miejscowego oświetlenia obiektów w scenie, podobnie jak fotograf naświetla lampą błyskową postacie na zdjęciu. Aby uzyskać efekt reflektora przy naświetlaniu obiektów należy w definicji źródła światła dodać słowo kluczowe spotlight.
- Reflektory charakteryzują się takimi parametrami jak:
  - radius (kąt nachylenia krawędzi bocznej stożka światła)
  - falloff (kąt stożka w miejscu rozpadu (wygaszenia) promieni światła)
  - tightness (wysokość falloff)
  - point_off (punkt padania światła)

Przykład

```
//przykład wcześniejszy
……
light_source
{
        <0, 10, -3>
        color White
        spotlight
        radius 15
        falloff 20
        tightness 10
        point_at <0, 0, 0>
}
```



Cylindryczne źródło światła

- Reflektory są źródłem światła o ukształtowaniu stożkowym. Intensywność światła reflektora zmienia się wraz z odległością. W celu zwiększenia zasięgu światła reflektora należy podnosić wartości parametrów radius i falloff. Czasami trzeba jednak, aby przy zachowaniu szczególnych wartości tych parametrów oświetlić obiekty bez względu na ich oddalenie od źródła światła.
- Problem ten rozwiązuje cylindryczne źródło światła. Po jego zastosowaniu uzyskamy efekt podobny jak przy reflektorach z tą zaletą, że wartości radius i falloff nie mają tak dużego wpływu na oświetlanie odległych obiektów. Kształt wiązki światła jest cylindrem (walcem).
- Cylindryczne źródło światła tworzy się analogicznie jak reflektory, zastępując słowo kluczowe spotlight słowem cylinder.

Obszary świetlne

Obszary świetlne (area_light)

Postać ogólna:

```
light_source
{
    <x, y, z>
    color rgb <r, g, b>
    area_light <x', y', z'>, <x", y", z">, m, n
    adaptive p
    jitter
}
```

gdzie:

<x, y, z> – wektor położenia źródła światła,

<r, g, b> – wektor koloru źródła światła,

<x', y', z'>, <x", y", z"> - wektory położenia obszaru światła,

m – ilość świateł w obszarze świetlnym leżących wzdłuż osi zdefiniowanej wektorem <x', y', z'>,

n – ilość świateł w obszarze świetlnym leżących wzdłuż osi zdefiniowanej wektorem <x", y", z">,

p – wartość parametru adaptive.

Obszary świetlne

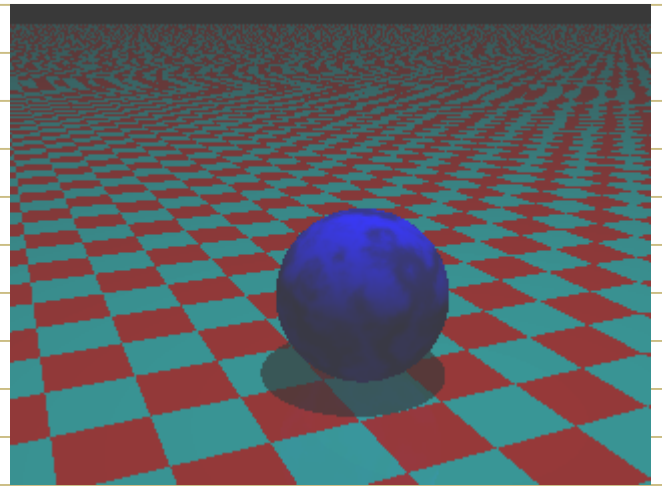
- Wszystkie dotychczas omówione typy źródeł światła miały jedną wspólną cechę – obiekty przez nie oświetlane rzucały ostre i wyraźne cienie. Działo się tak dlatego, ponieważ źródłem światła był nieskończenie mały punkt. W świecie rzeczywistym światło oświetlające obiekty nie jest nieskończenie małym punktem, ale posiada pewien wymiar. Cienie obiektów są wtedy bardziej miękkie i mniej widoczne.
- Obszar świetlny jest strukturą dwuwymiarową, więc jego rozmiar definiujemy za pomocą dwóch wektorów, określających ile jednostek wzdłuż danej osi zajmuje obszar. Następnie należy określić, ile świateł ma występować w szyku na obszarze świetlnym. Im więcej świateł, tym cienie są bardziej miękkie, lecz czas renderingu znacznie się wydłuża.

Obszary świetlne

- Area_light posiada dwa atrybuty: adaptive i jitter.
- Atrybut adaptive nakazuję programowi dostosowywanie się do sytuacji i przesyłanie tylko promieni potrzebnych do wygenerowania pikseli. Jeżeli parametr adaptive zostanie pominięty, dla każdego światła w obszarze świetlnym będzie przesyłany oddzielny promień, co może wydłużyć czas śledzenia.
- Atrybut jitter nakazuje nieznaczne zmiany pozycji każdego światła w obszarze świetlnym tak, aby cienie oświetlanych obiektów były miękkie i naturalne.

Przykład

```
// tak jak wcześniej
sphere
{
    <0,0,0>,1 texture { Sapphire_Agate
    }
    translate <1.5, 0, -2>
}
light_source
{
    <2, 10, -3>
    color White
    area_light <5, 0, 0>, <0, 0, 5>, 5, 5
    adaptive 1
    jitter
}
```



Obiekty

Postać ogólna:

nazwa_obiektu

{

<wektor_położenia>,

pokrycie_obiektu (kolor i struktura powierzchni)

}

Obiekty

Płaszczyznę definiujemy poprzez podanie wektora normalnego do płaszczyzny i odległości płaszczyzny od środka układu współrzędnych.

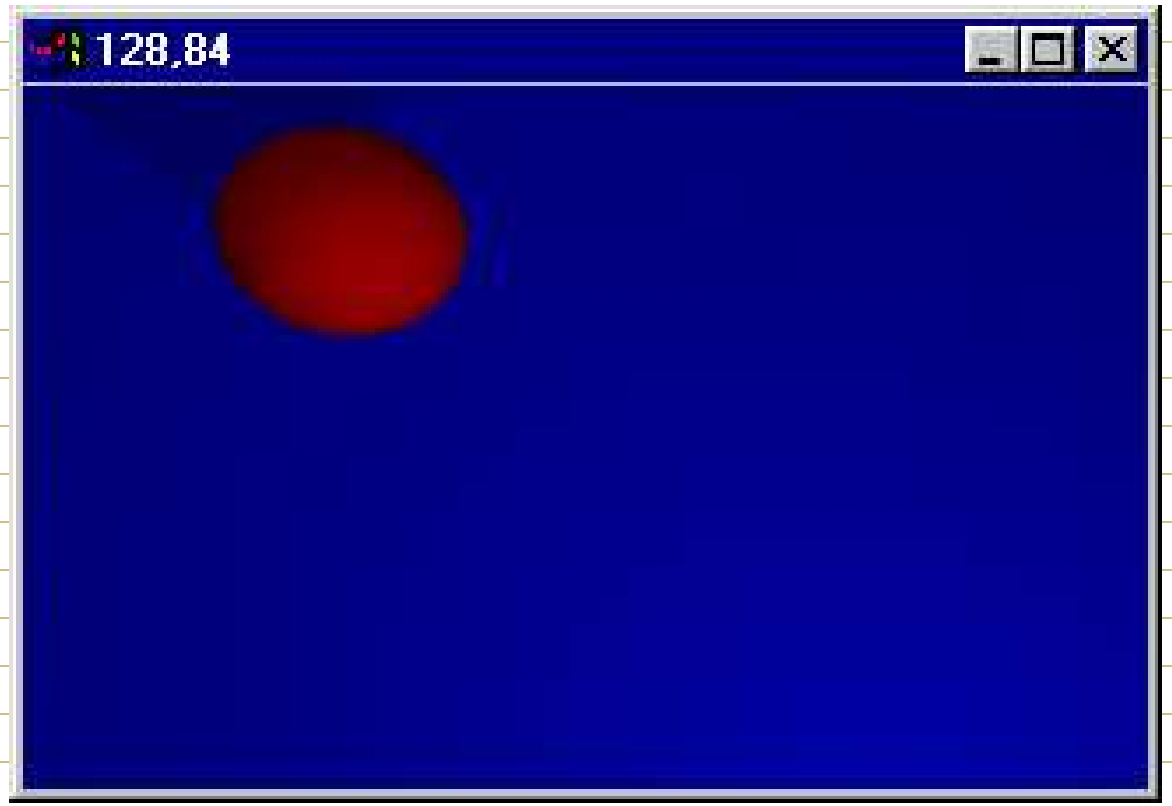
PLANE:

plane { <Normal>, Distance [OBJECT_MODIFIERS...] }

Przykład:

plane{ <0,1,0>, -1 texture{pigment{Blue}}}

Przykład



Obiekty

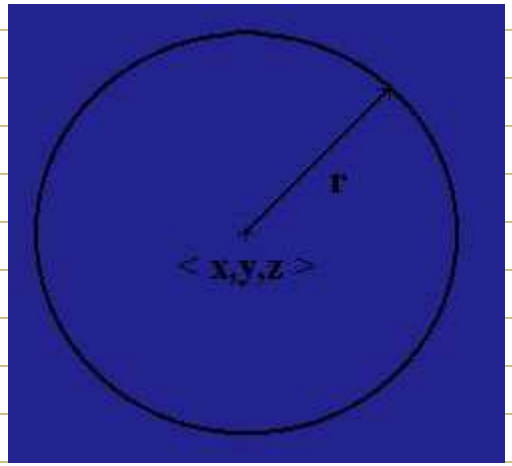
Kula jest definiowana poprzez współrzędne środka i promień

sphere{ <-2,0,2> 1 texture{pigment{Red}}}

SPHERE:

sphere { <Center>, Radius
[OBJECT_MODIFIERS...] }

Sfera



Obiekty

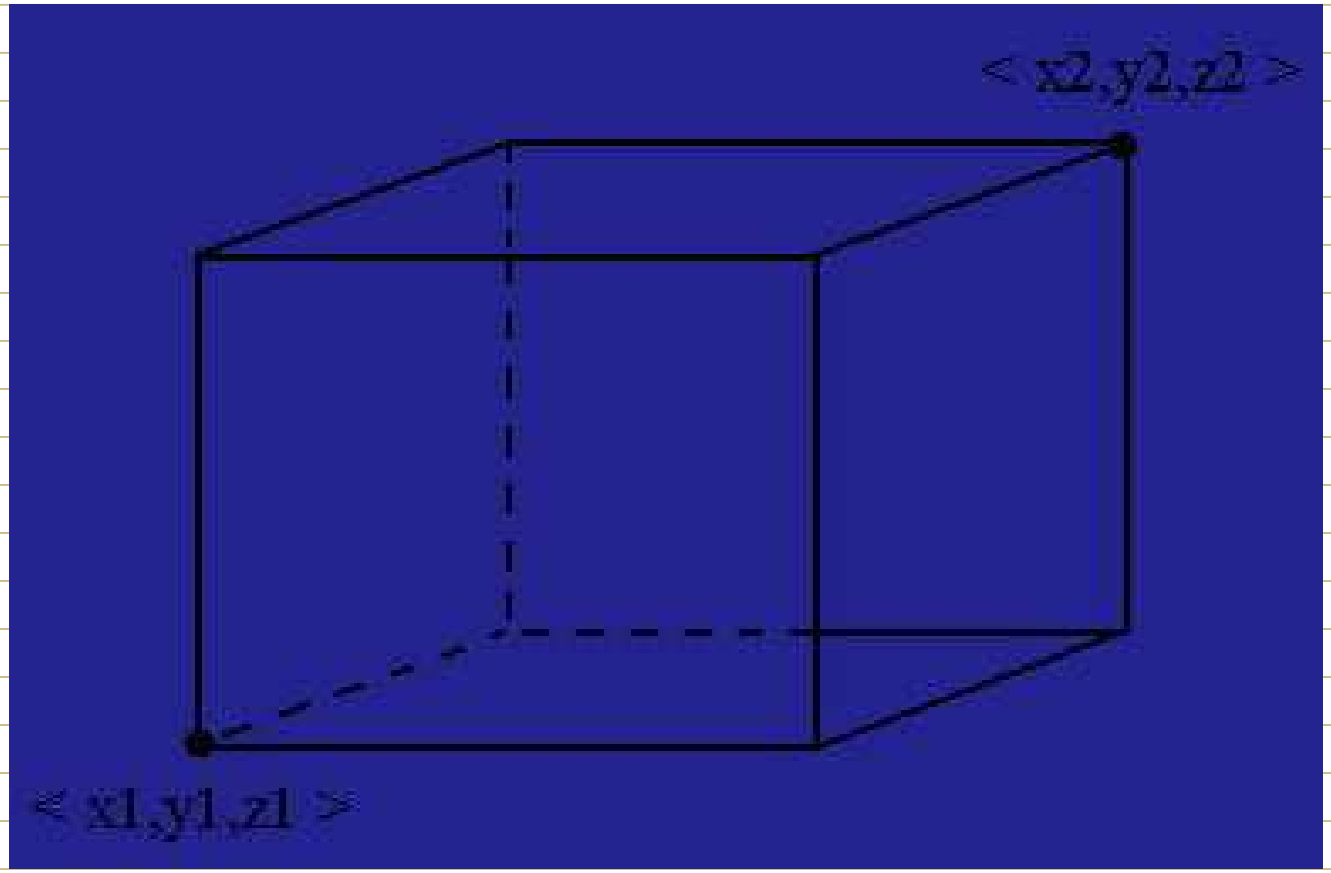
- Pudełko definiujemy przez podanie współrzędnych przeciwległych wierzchołków, ściany są równoległe do płaszczyzn wyznaczanych przez osie układu współrzędnych.

  BOX:

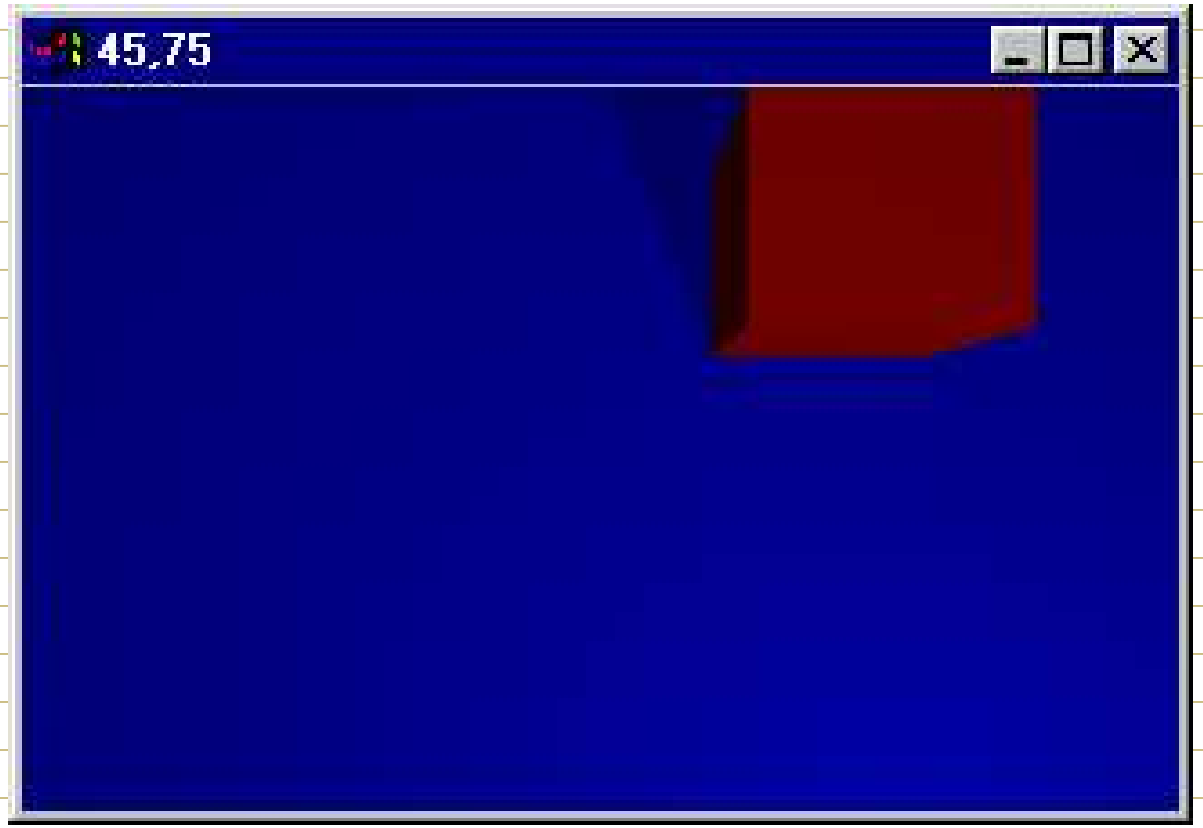
  box { <Corner_1>, <Corner_2>
  OBJECT_MODIFIERS...]}

- box{<1,-1,3> <3,1,1>texture{pigment{Red}}}

Pudełko



Przykład



Obiekty

Torus: środek torusa leży w środku układu współrzędnych w płaszczyźnie x-z.

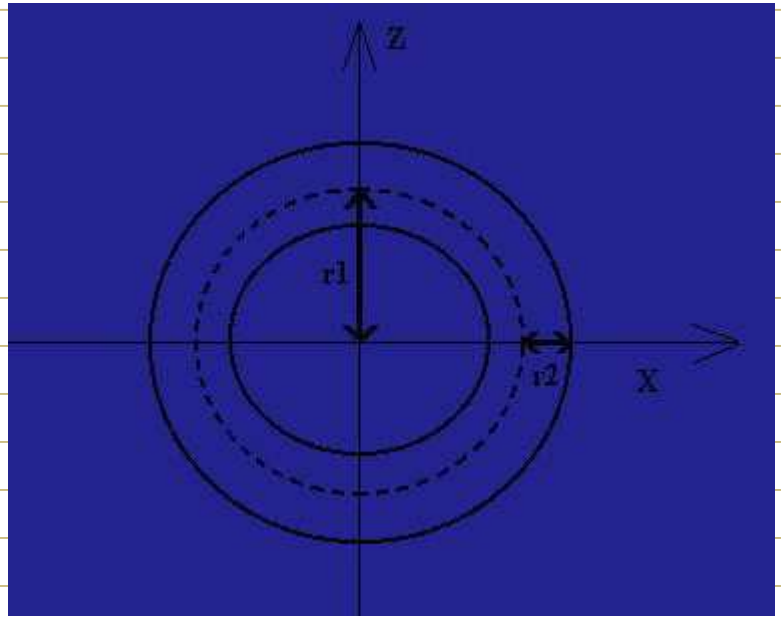
Definiowany jest poprzez dwa promienie: całego torusa i "oponki"

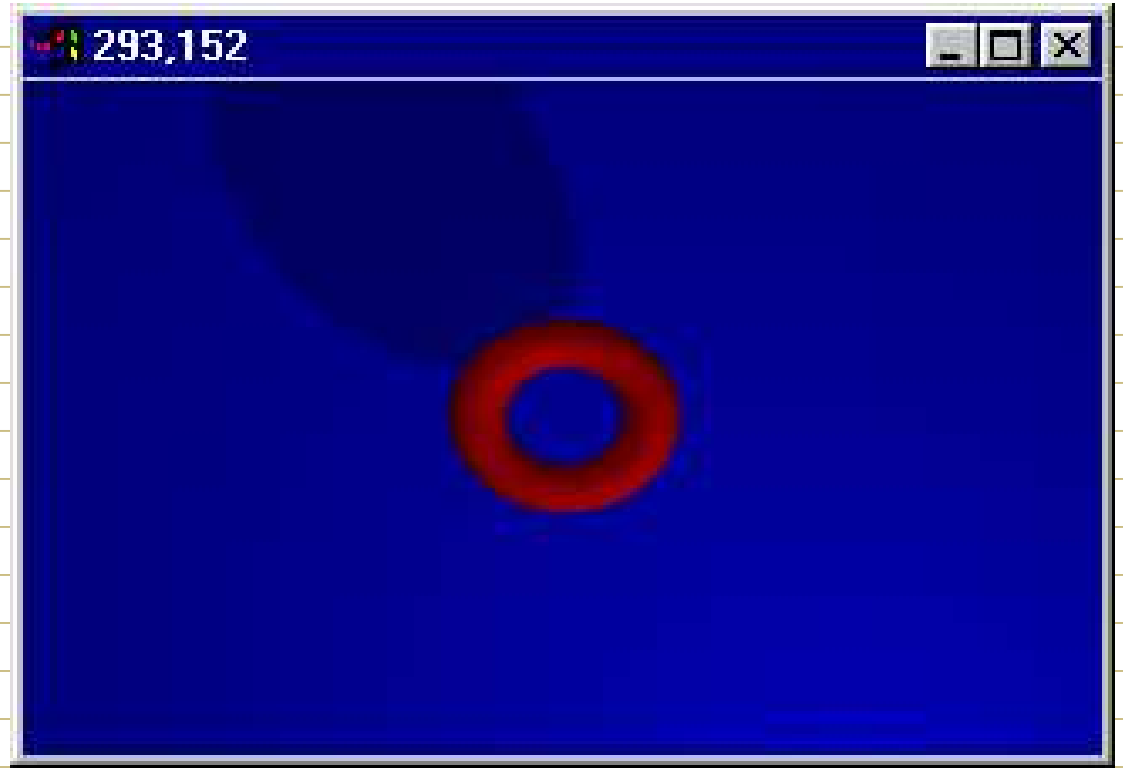
torus{0.75, .25texture{pigment{Red}}}

TORUS:

torus { Major, Minor}

Torus




293,152

Obiekty

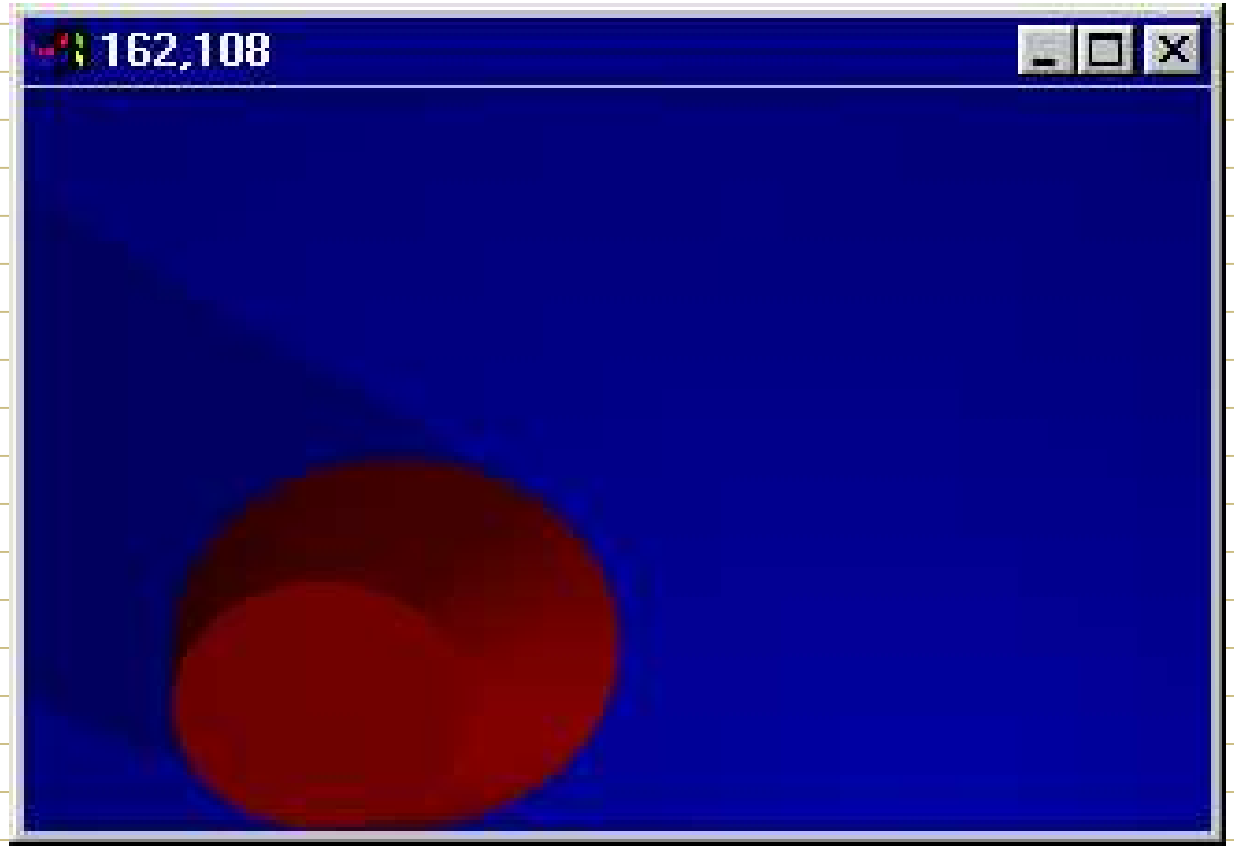
- Cylinder jest wyznaczany przez środki i promienie podstaw stożków

```
cone{ <-2,-1,-2>,1   <-2,1,-2>,1
   texture{pigment{Red}}}
cone{ < 2,-1,-2>,1.2 < 2,1,-2>,0
   texture{pigment{Red}}}
```

CONE:

cone { <Base_Point>, Base_Radius, <Cap_Point>, Cap_Radius}

Przykład



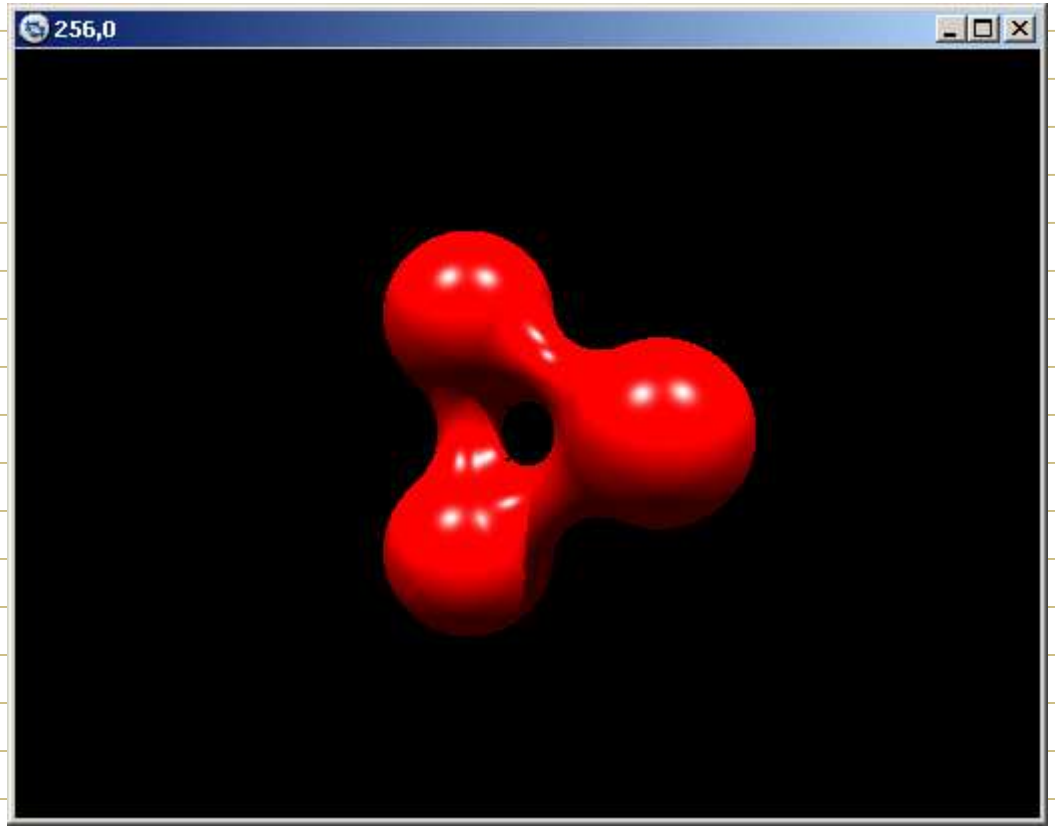
Obiekty

- Postać ogólna:
- **blob**
- **{**
- **threshold n**
- **objekt1 { definicja_obiektu1 }**
- **objekt2 { definicja_obiektu2 }**
- **}**
- gdzie:
- n – wartość parametru threshold.
- Blob jest bardzo interesującym typem obiektów. Tworzony jest z kul i cylindrów (walców), których połączenia stanowią wygładzone powierzchnie.

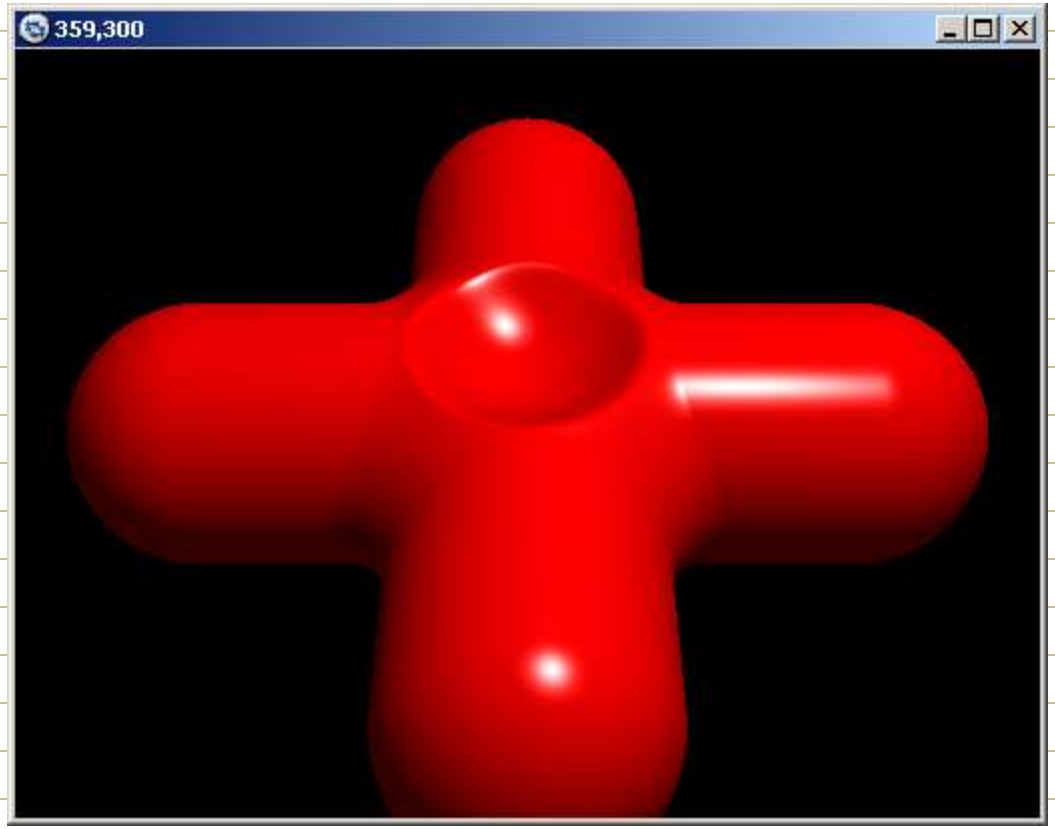
Obiekty

- Według matematycznego opisu blob'y są obiektami, których powierzchnia jest definiowana przez siłę pól w każdym jej punkcie. Siła ta osiąga wartość maksymalną w środku obiektu i opada do zera na powierzchni obiektu. Pola siłowe we wszystkich komponentach (obiektach składowych) są dodawane i formują pole siłowe obiektu. Następnie program rozpatruje punkty, gdzie pole siłowe ma wartość progową. Punkty te kształtują powierzchnię blob'u. Wszystkie punkty z większą wartością pola od wartości progowej są uważane za zewnętrzne, nie należące do obiektu, natomiast wszystkie punkty, których wartości pól siłowych są mniejsze od wartości progowej są uznane za wewnętrzne, tworzące obiekt.

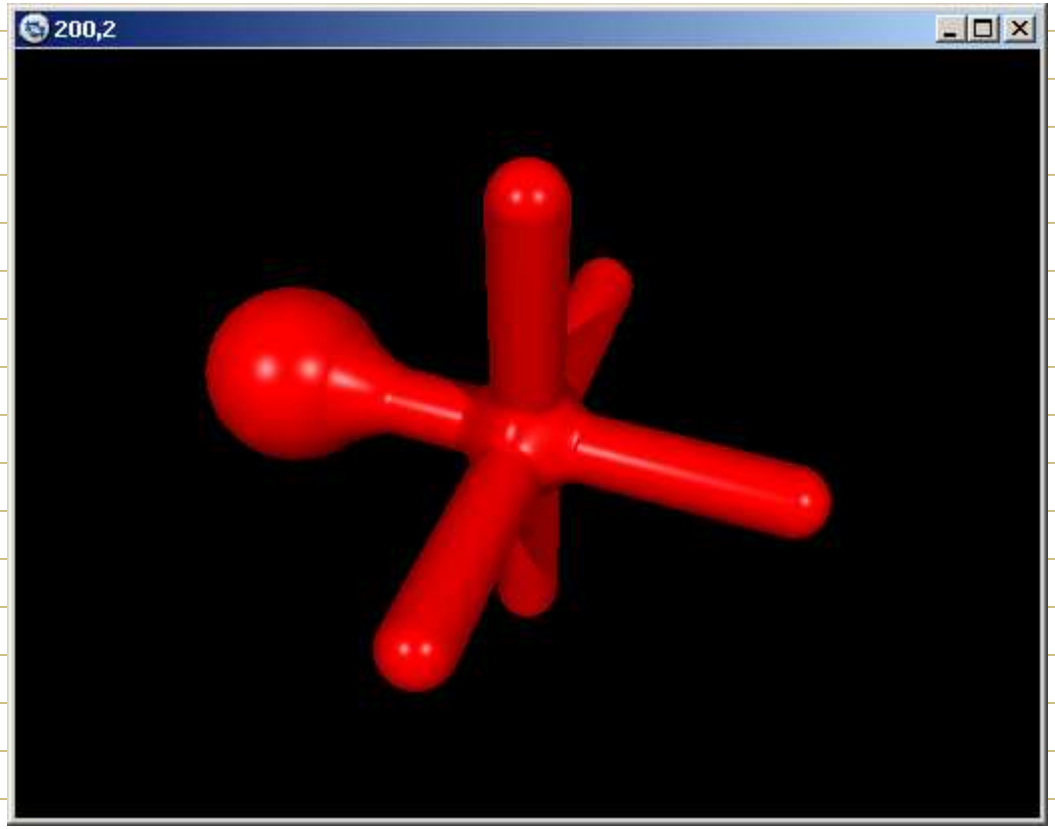
Przykład1



Przykład2



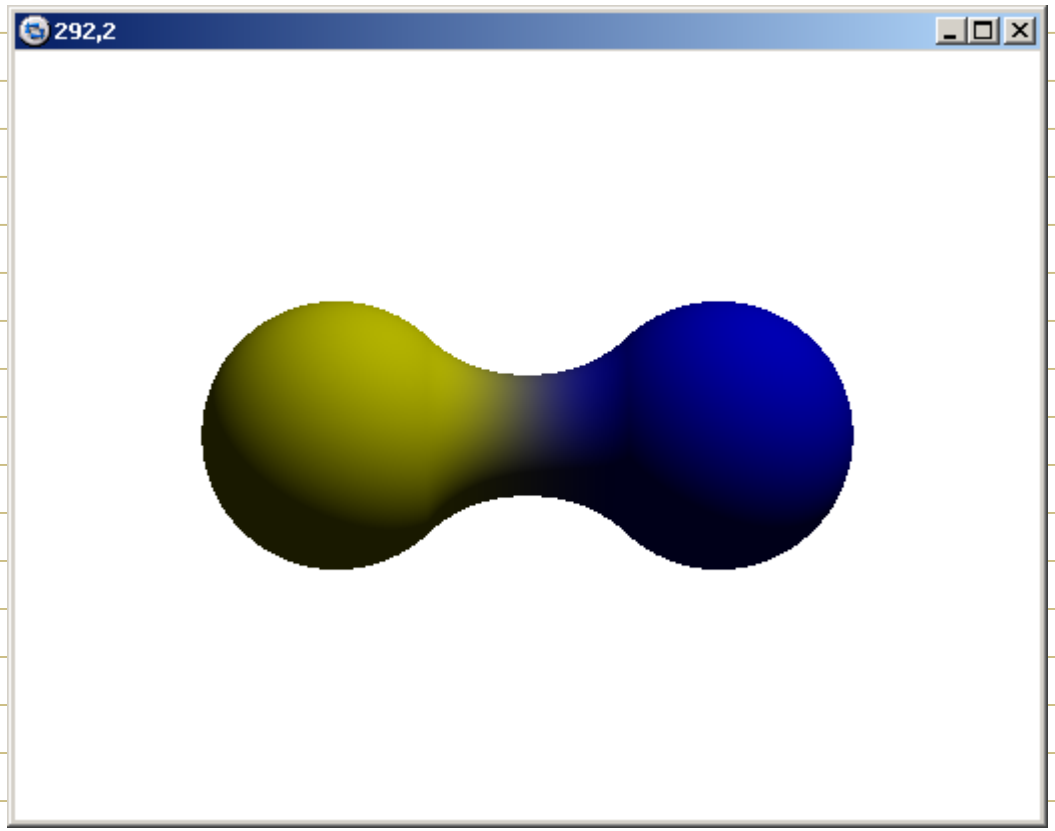
Przykład3



Przykład

```
#include "colors.inc"
background{White}
camera
{angle 15
location <0,2,-10>
look_at <0,0,0>}
light_source { <10, 20, -10> color White }
blob
{threshold .65
sphere
{ <.5,0,0>, .8, 1
 pigment {Blue}}
sphere
{<-.5,0,0>,.8, 1
pigment {Yellow}}}
```

Wynik



Przykład

```
#include "colors.inc"
#include "glass.inc"
#include "woods.inc"
camera {angle 15 location <0,2,-10> look_at 0}
light_source {<10,20,-10> color White}
light_source {-2 color White spotlight point_at 0 radius 100}
plane {<0,1,0>, -15 texture {pigment {White*2}}}
union{blob {threshold 0.65
sphere {<0.5,0,0>, 0.8, 1 texture {T_Glass4} finish {refraction 0}} //światło przechodzi bez ugięć
sphere {<-0.5,0,0>, 0.8, 1 texture {T_Glass4} finish {refraction 0}}
rotate <0,0,90>}
torus {0.55,0.1 texture {T_Wood7} translate <0,0.8,0>}
torus {0.55,0.1 texture {T_Wood7} translate <0,-0.8,0>}
cylinder {<0,0.7,0>, <0,0.9,0>, 0.55 texture {T_Wood7}}
cylinder {<0,-0.7,0>, <0,-0.9,0>, 0.55 texture {T_Wood7}}
cylinder {<-0.35,-0.85,-0.4>, <-0.35,0.85,-0.4>, 0.06 texture {T_Wood7}}
cylinder {<-0.35,-0.85,0.4>, <-0.35,0.85,0.4>, 0.06 texture {T_Wood7}}
cylinder {<0.35,-0.85,0.4>, <0.35,0.85,0.4>, 0.06 texture {T_Wood7}}
cylinder {<0.35,-0.85,-0.4>, <0.35,0.85,-0.4>, 0.06 texture {T_Wood7}}
sphere {<0.006,-0.48,-0.03>, 0.28 scale <1.26,1,1> pigment {color Blue}}
rotate <12,-85,12>}
text {ttf "timrom.ttf", "czas ucieka", 0.01, <0,0,0> scale 0.25 translate <-0.5,0.6,0> rotate <10,10,-
65>}
```

Wynik

Obiekty

- Tło
  - background{kolor}

Deklarowanie własnych obiektów

- Postać ogólna:
- **#declare nazwa_obiektu=**
- **{**
- **definicja_obiektu**
- **}**
- Język opisu scen dla POV-Ray'a umożliwia deklarowanie własnych obiektów. Służy do tego słowo kluczowe declare, po którym następuje nazwa a następnie opis obiektu.
- **Przykład 7.** Deklaracja własnego obiektu
- `#declare wlasny =`
- `sphere`

Wstępnie zdefiniowane materiały pokrycia obiektów

- Program POV-Ray™ posiada wiele plików wsadowych, które zawierają definicje struktur powierzchni obiektów. Aby wykorzystać zdefiniowany materiał pokrycia obiektów, należy umieścić w nagłówku pliku z opisem sceny nazwy plików zawierających definicje. Częściej stosowane to:
- - glass.inc (struktura szklana),
- - metals.inc (struktura metalowa),
- - stones.inc (struktura kamienna),
- - woods.inc (struktura drewniana).

Parametr pigment

- Postać ogólna:
- **texture**
- **{**
- **pigment {kolor}**
- **}**
- Parametr pigment jest częścią polecenia texture. Definiuje on pokrycie obiektu odpowiednim kolorem.

Mapy kolorów

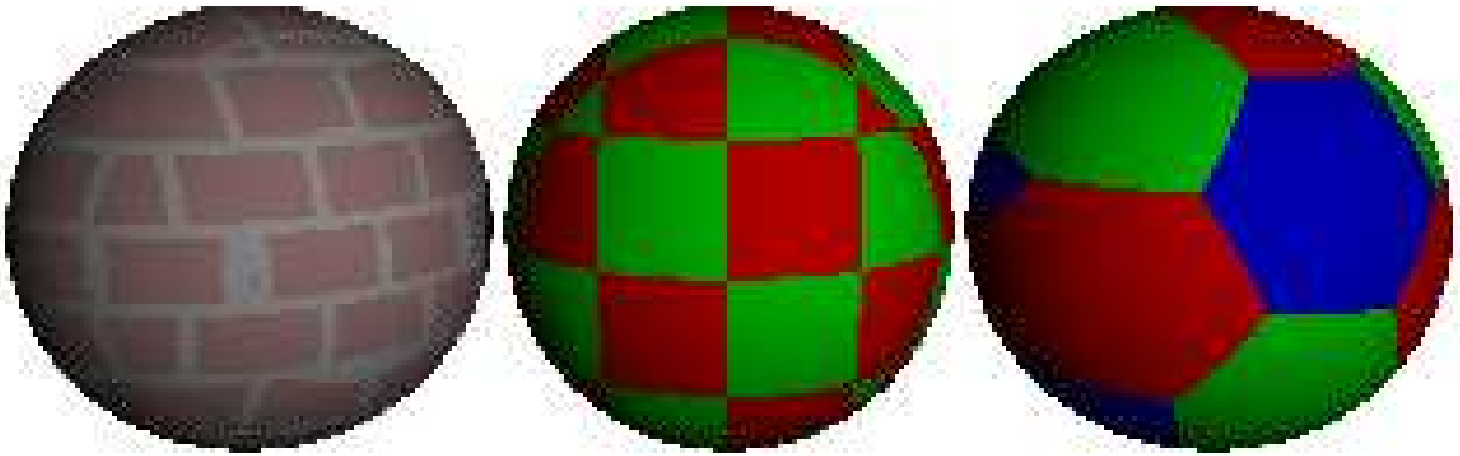
- Postać ogólna:
- color_map
- {
- [kolor1]
- [kolor2]
- [kolor3]
- }
- Oprócz pojedynczych kolorów do pokrycia obiektów w języku POV-Ray'a możliwe jest zdefiniowanie mapy kolorów. Służy do tego słowo kluczowe color_map, definiujące grupy kolorów na "mapie" w zakresie od 0.0 do 1.0

Pigmenty ze względu na zarządzanie grupami kolorów można podzielić na kilka rodzajów, m.in.: gradient, turbulence, marble, bozo.

Przykład . Zastosowanie jako barwy kuli zestawień kolorów **brick, checker i hexagon**

```
#include "colors.inc"
background {White}
camera
{location <0, 2, -3> look_at <0, 1.5, 0>}
sphere {  <-2.1, 1, 2>, 1
   texture {  pigment {brick Gray75, Pink}       scale 0.1
  rotate 45*y      } }
sphere {  <0, 1, 2>, 1
   texture {       pigment {checker Yellow Red}
       scale 0.5      } }
sphere  {    <2.1, 1, 2>, 1
   texture {
       pigment {hexagon Green Red Blue}
       scale 0.5
       rotate 90*x     } }
light_source {<2, 3, -4> White}
```

Wynik



Pigmenty: "gradient”, "turbulence„, "marble” i parametr "bozo"

- Pigment gradient definiuje pokrycie obiektu grupami kolorów bazując na mapie kolorów. Wektor następujący po słowie kluczowym gradient jest standardowym wektorem służącym do określenia orientacji grupy kolorów.
- Pigment turbulence "miesza" materiały (np. mapy kolorów) jakimi pokryty jest obiekt. Efekt uzyskany po zastosowaniu pigmentu turbulence przypomina plazmatyczne plamy utworzone na fundamencie mapy kolorów. Pigment turbulence przyjmuje wartości od 0.0 do 1.0. Słowo kluczowe turbulence musi występować zawsze po instrukcji gradient, która również i w tym przypadku określa orientację grupy kolorów zdefiniowanych do pokrycia obiektu.
- Pigment marble umożliwia pokrycie kuli strukturą marmurkową. Podobnie, jak w poprzednich przypadkach struktura pokrywająca obiekt utworzona jest z mapy kolorów.
- Kolejnym parametrem zarządzającym mapami kolorów jest parametr bozo. Parametr posiada słowo kluczowe frequency odpowiadające za częstotliwość występowania mapy kolorów na jednostkę rozmiaru.

Przykład. Zastosowanie pigmentów gradient, turbulence, marble, bozo

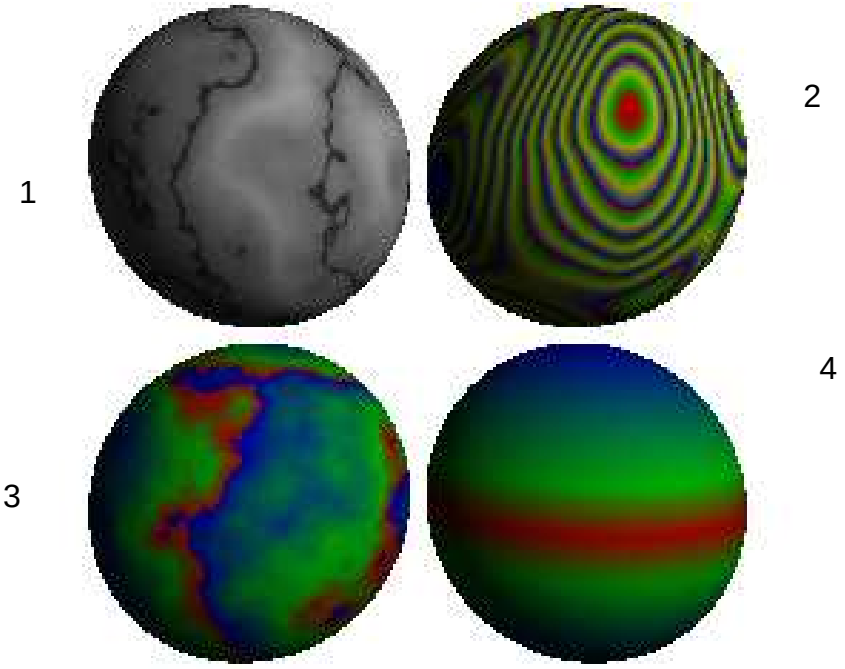
```
#include "colors.inc"
background {White}
camera
{
location <-1, 1, -4.5>
look_at <-1, 1, 0>
}
sphere
{<-2.1, 2.1, 0>, 1
  pigment {
    marble
    turbulence 1
    color_map{
              [0.0 Gray90]
              [0.9 Gray60]
              [1.0 Gray20]
    }   }}
sphere
{<-2.1, 0, 0>, 1
  pigment {
    gradient <1,0,0>
    turbulence 1
    color_map{
              [0.0 Red]
              [0.25 Green]
              [1.0 Blue]
    }  } }
sphere
{
<0, 0, 0>, 1
     pigment {
       gradient <0,1,0>
       color_map{
                 [0.0 Red]
                 [0.25 Green]
                 [1.0 Blue]
       }      } }

sphere {
<0, 2.1, 0>, 1

       pigment {
       bozo
       frequency 10
       color_map{
                 [0.0 Red]
                 [0.25 Green]
                 [0.50 Yellow]
                 [1.0 Blue]
                 }  }       }
light_source {<2, 3, -5> White}
```

Wynik

- marble (1), bozo (2), turbulence (3) i gradient (4).



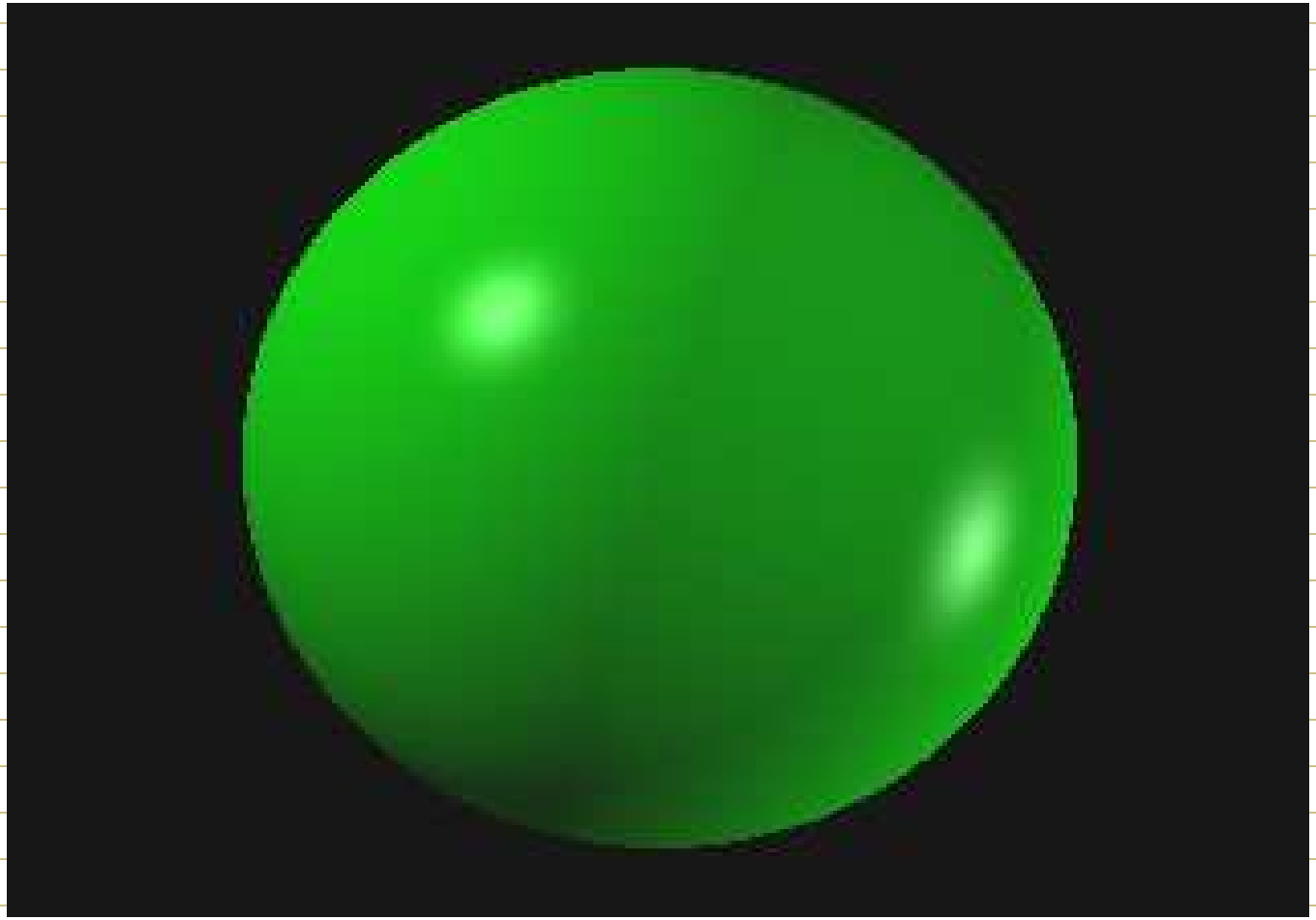
Parametry finish i phong

- Parametr finish opisuje, w jaki sposób dany obiekt reaguje na światło: jak dużo refleksów (odbić) świetlnych, stopień połysku, metaliczność obiektu, odbicie otoczenia itp. Wszystkie cechy parametru finish zamknięte są w nawiasach {}.
- Parametr phong definiuje podświetlenie lub blask czyli tzw. połysk obiektu. Przyjmuje wartości od 0.0 do 1.0, które określają, jak jasny jest efekt. Dodaje realistycznych błysków w miejscach "uderzania" promieni świetlnych biegnących od źródeł światła.

Przykład. Zastosowanie efektu phong

```
#include "colors.inc"
camera
{
location <0, 2, -3>
look_at <0, 1, 2>
}
sphere
{<0, 1, 2>, 2
texture {
      pigment {Green}
      finish {phong 0.4}
    }}
light_source {<-5, 7, -3> White}
light_source {<55, -15, -3> White}
```

Wynik



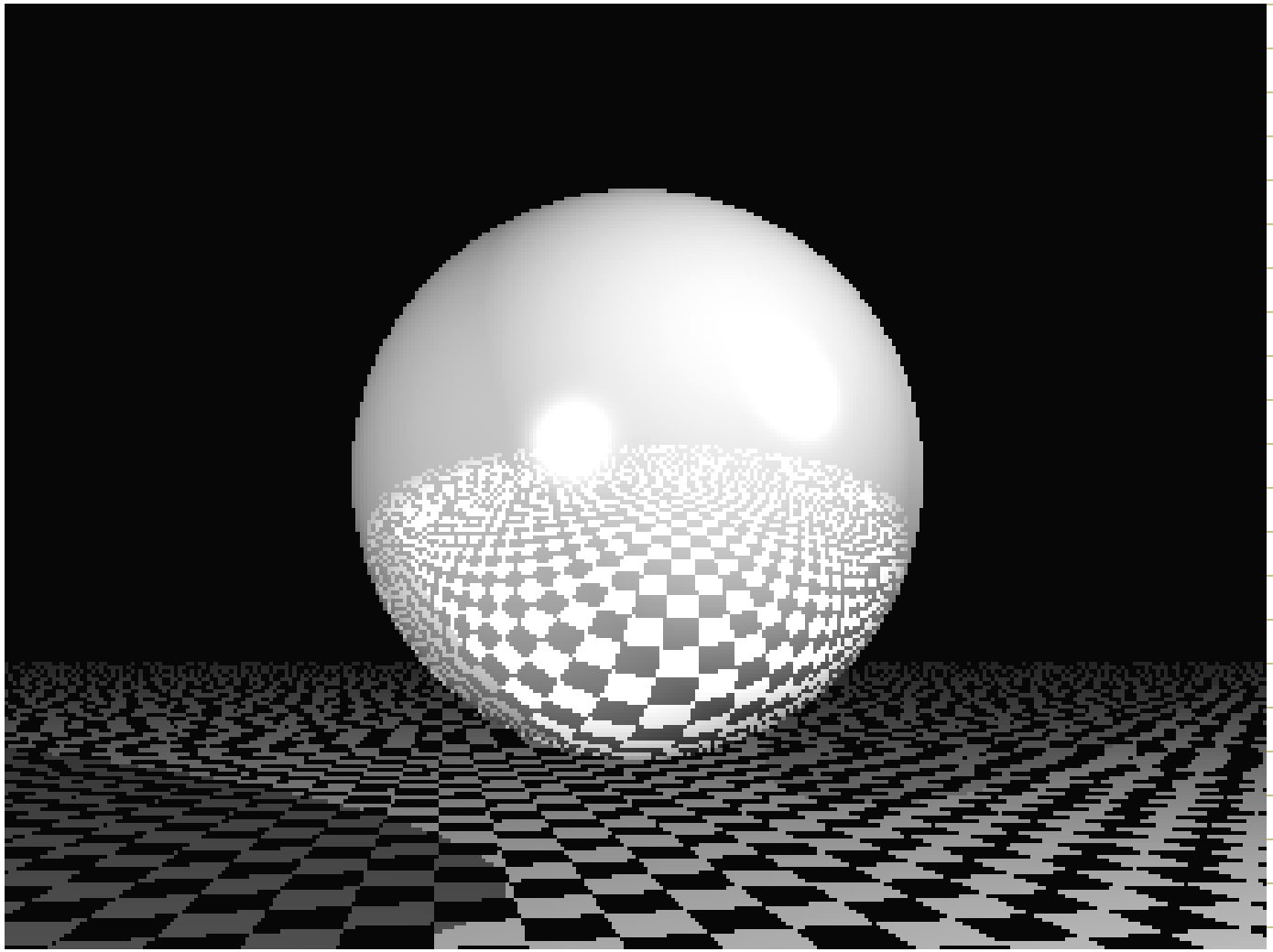
Parametr "reflection"

Parametr reflection nadaje obiektowi pokrycie zwierciadlane. Przyjmuje wartości w zakresie od 0.0 do 1.0, które określają w jakim stopniu obiekt odbija otoczenie.

Przykład. Zastosowanie efektu reflection

```
include "colors.inc"
camera {
location <-2, 3, -10>
look_at <0, 5, 0>
}
plane
{<0, 1, 0>, 0
    pigment { checker color <0,0,0> color <1,1,1> }}
sphere
{<0, 5, 0>, 3
texture {
      pigment {White}
      finish {
      reflection 0.9
      phong 0.5
      }     } }
light_source {<10, 10, -10> White}
```

Wynik



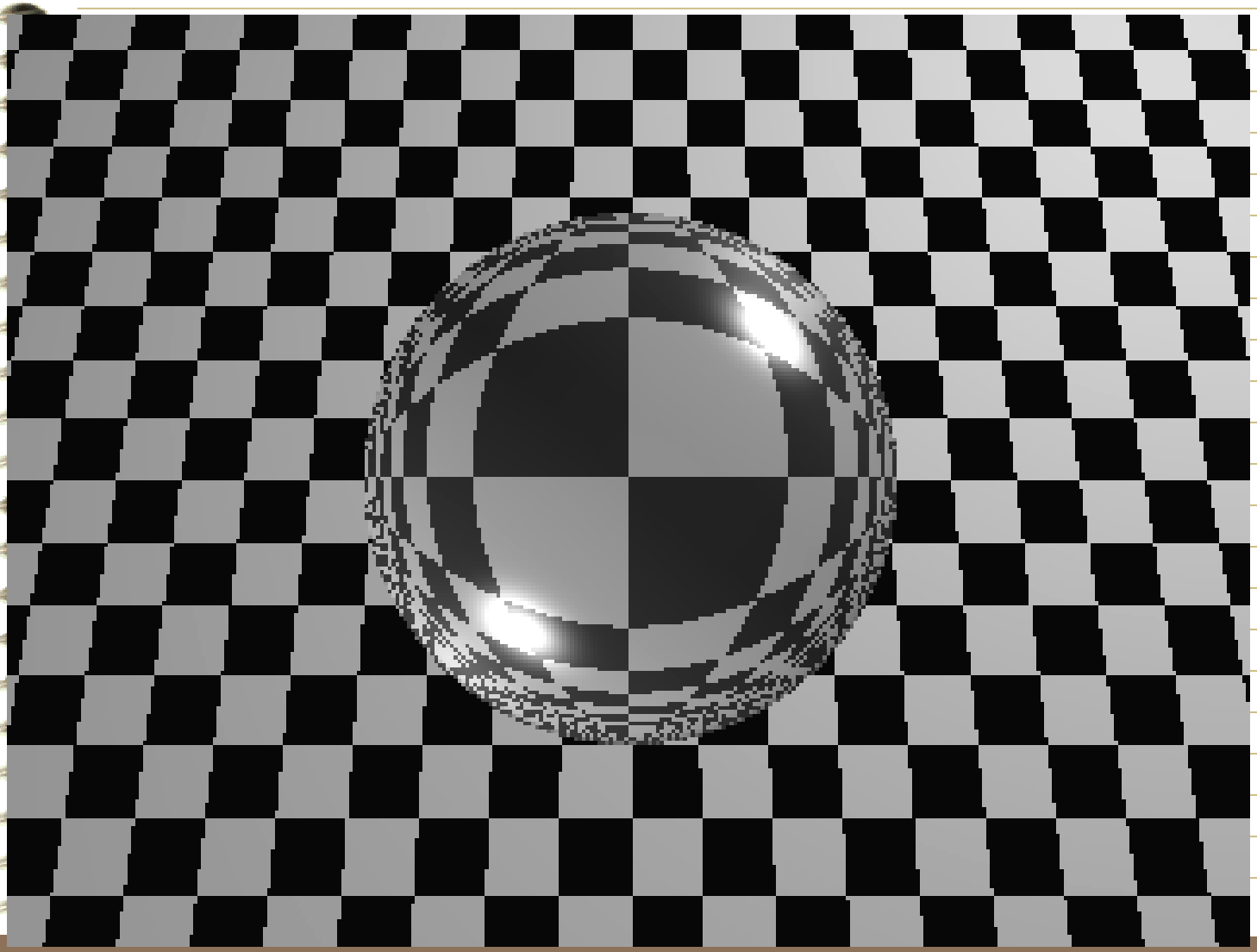
Parametr "refraction"

Parametr refraction definiuje realistyczne załamanie promieni świetlnych przechodzących przez przezroczysty obiekt. Przyjmuje wartości 0.0 – obiekt nieprzezroczysty i 1.0 – przezroczysty. Używanie wartości pośrednich nie jest zalecane ze względu na niezgodne z rzeczywistością załamanie promieni. Stopień widoczności zniekształceń kontroluje słowo kluczowe ior (ang. *index of refraction*), przyjmujące wartości większe od zera. Wartość domyślna ior dla "pustej przestrzeni" wynosi 1.0.

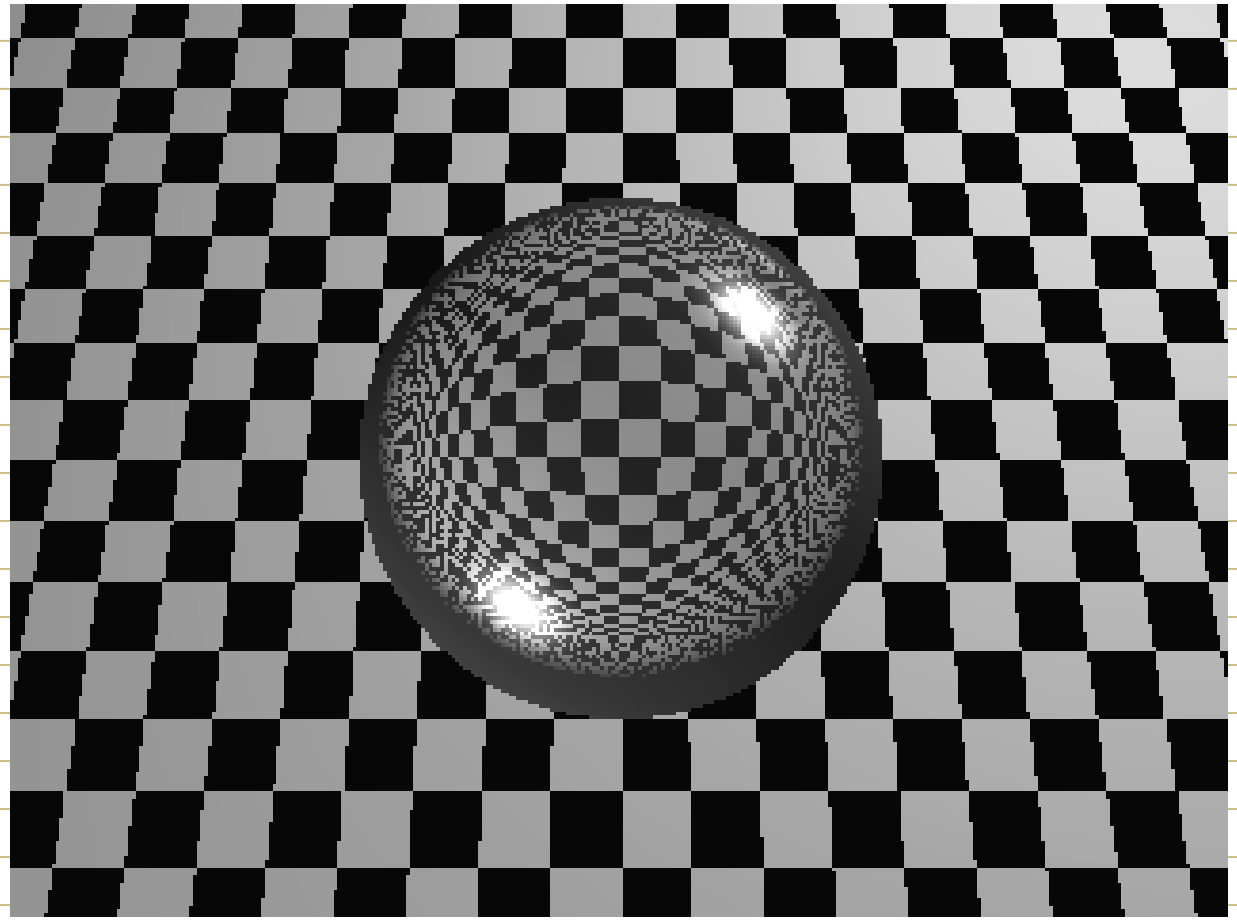
Przykład. Zastosowanie efektu refraction dla ior=1,5

```
#include "colors.inc"
camera
{location <-2, 12, 0>
look_at <0, 5, 0>}
plane
{<0, 1, 0>, -2
pigment { checker color <0,0,0> color <1,1,1> }}
sphere
{<0, 5, 0>, 2
texture {
        pigment { color rgbf <1,1,1,.8>}
        finish {
         refraction 1.0
         ior 1.5
         phong 1.0
       }     } }
light_source {<10, 10, -10> White}
light_source {<-30, 5, 20> White}
```

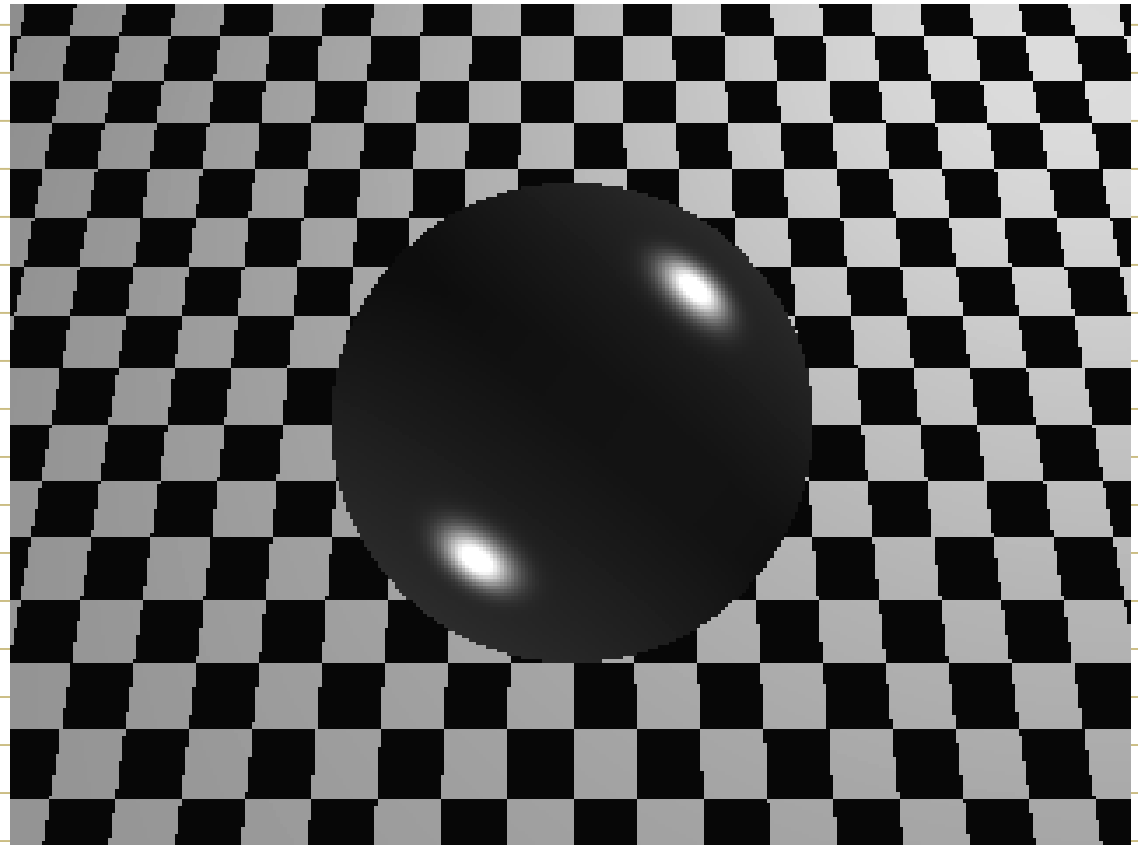
Wynik dla ior=1.5 i refraction=1



Wynik dla ior=3.5 i refraction=1



Wynik dla ior=3.5 i refraction=0



Parametr normal

Parametr normal tworzy na powierzchni pokrycia obiektu trójwymiarowe efekty: wyboje, fale, marszczenia itp.

Parametr "bumps"

Parametr bumps definiuje wyboistą strukturę pokrycia obiektu.

Parametr "ripples"

Po zastosowaniu parametru ripples można uzyskać efekt przypominający marszczenie się (falowanie) wody.

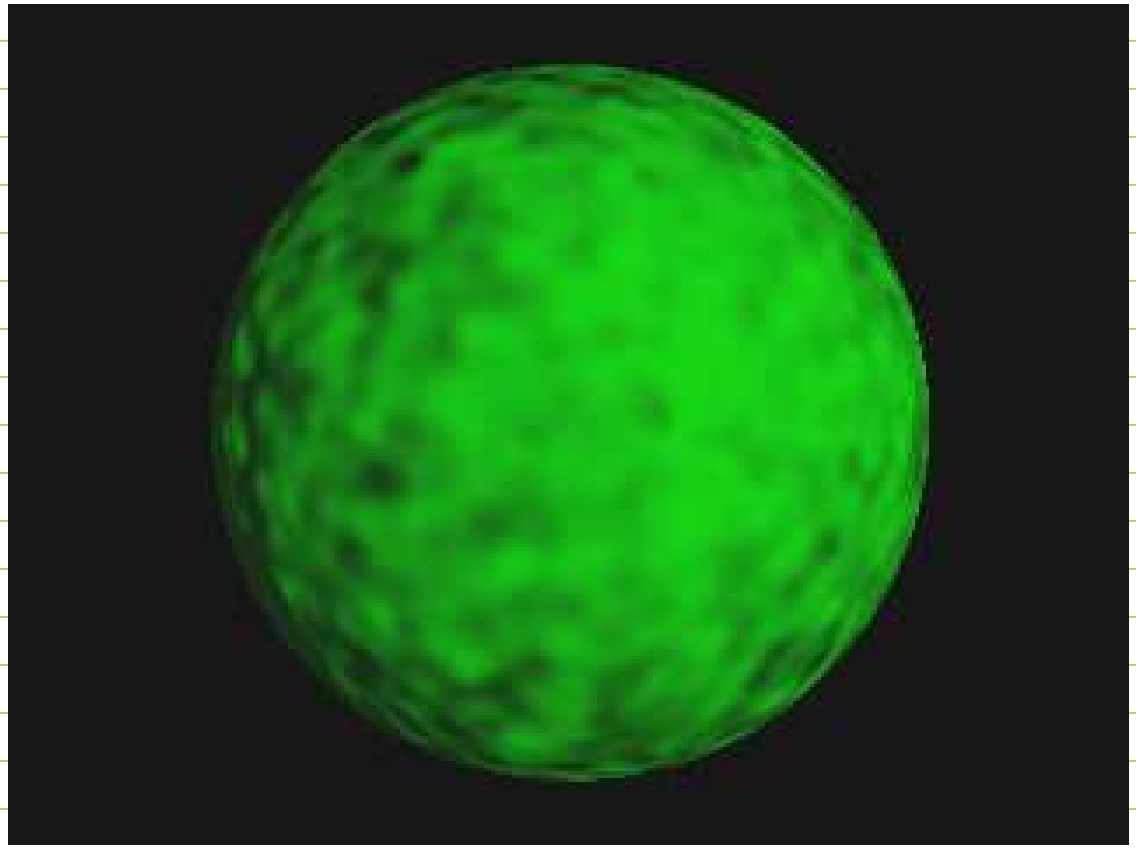
Parametr "waves"

Powierzchnia obiektu po zastosowaniu efektu waves przypomina rozmycie, poruszenie.

Przykład. Parametr bumps

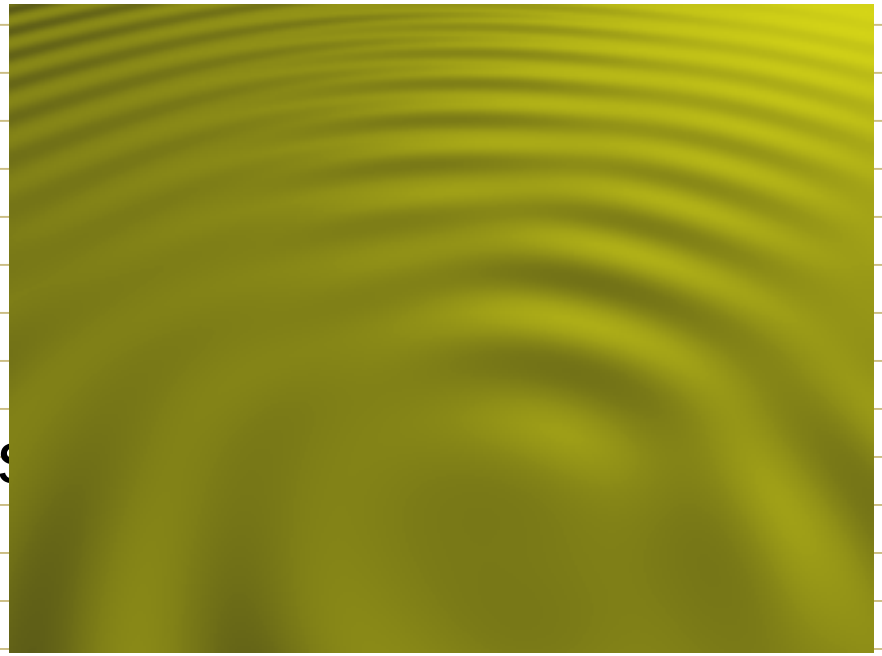
```
#include "colors.inc"
camera
{
location <0, 2, -3>
look_at <0, 1, 2>
}
sphere
{
<0, 1, 2>, 2
texture {
    normal {
     bumps 0.6
     scale 1/6
    }
  pigment {Green}
 }
}
light_source {<2, 3, -4> White}
```

Wynik

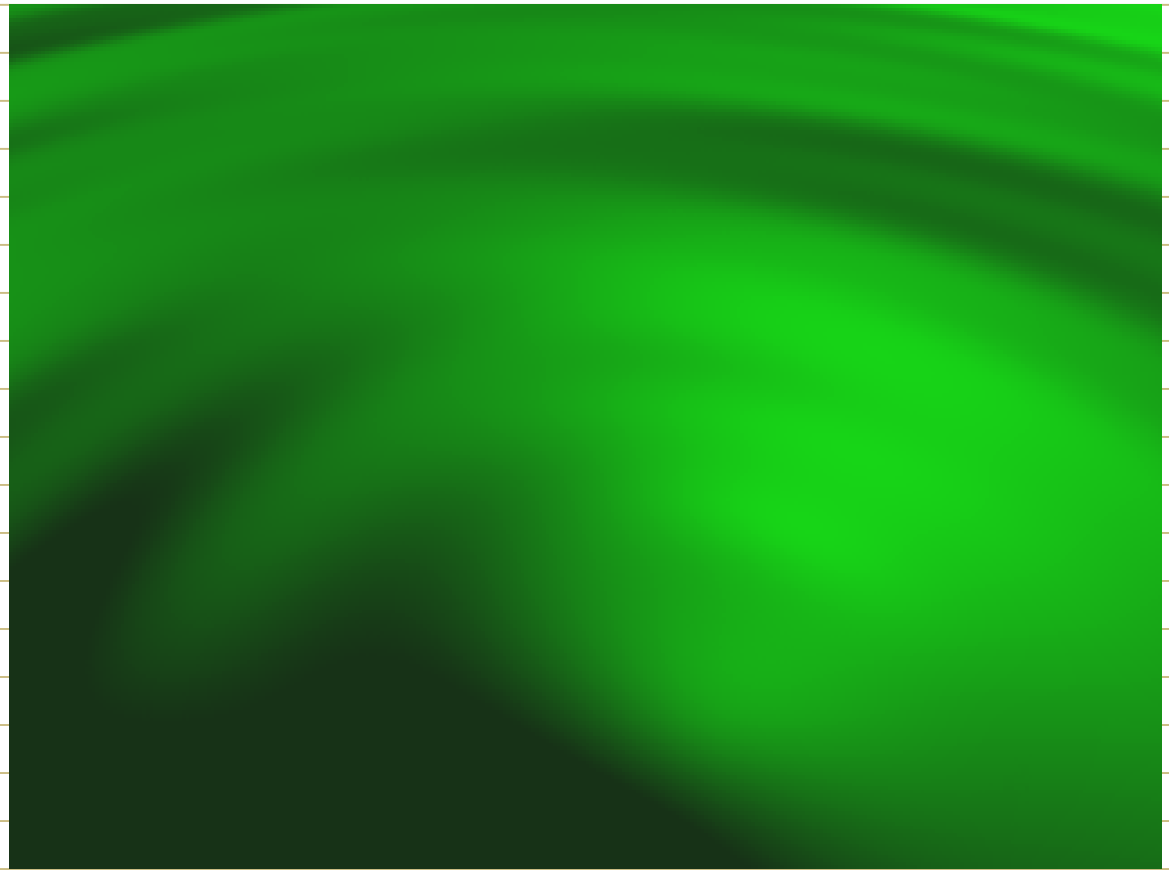


Parametr ripples

```
plane
{
<0, 1, 0>, -2
pigment {Yellow}
  normal {ripples
  0.8}
}
```



Parametr waves



Deklarowanie własnych struktur

Postać ogólna:

#declare nazwa_pigmentu =

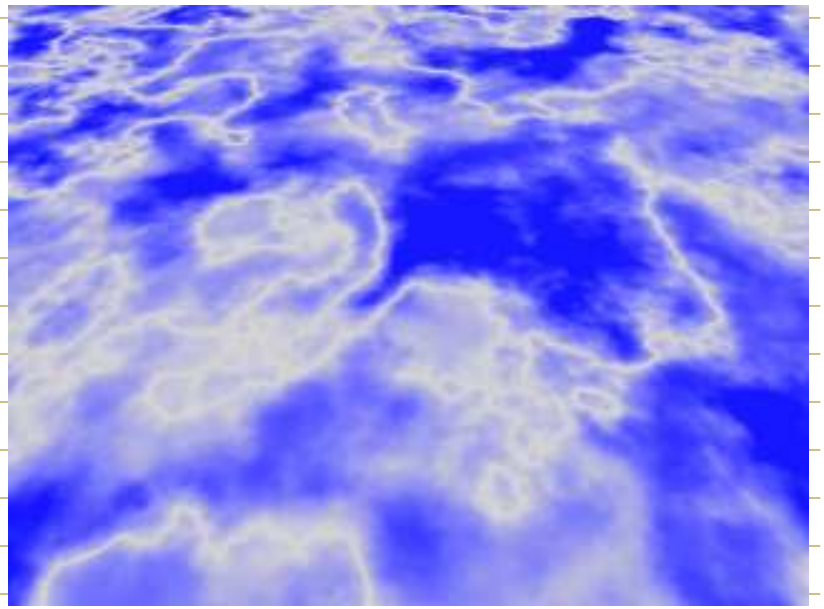
{

definicja_pigmentu

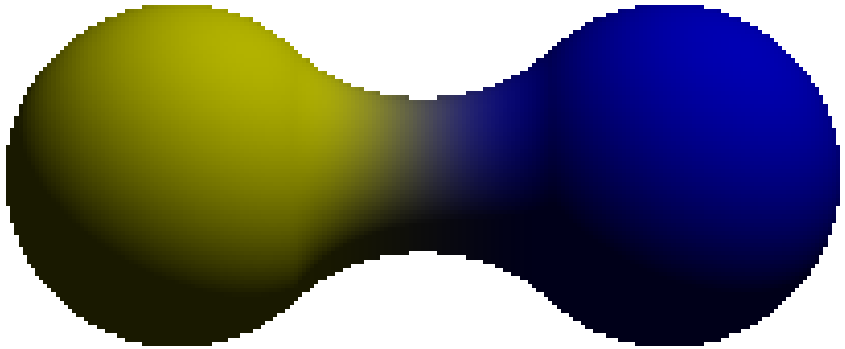
}

Przykład

```
#declare Chmury =
pigment
{
bozo
turbulence 1
color_map {
[0.0   color   White
  filter 1]
[0.5 color White]
[1.0   color   White
  filter 1]
}
}
```



Wynik



Efekty atmosferyczne

Język opisu scen POV-Ray'a umożliwia dodawanie do scen różnorodnych efektów atmosferycznych. Efekty te mają duży wpływ na wygląd obiektów w scenie oraz na właściwości świateł i cieni. Do najczęściej używanych efektów zaliczamy słońce, chmury, mgła, tęcza.

Sky sphere (obszar nieba)

Postać ogólna:

```
sky_sphere
{
pigment { parametr
color_map {
[kolor1}
[kolor2]
}
}
}
```

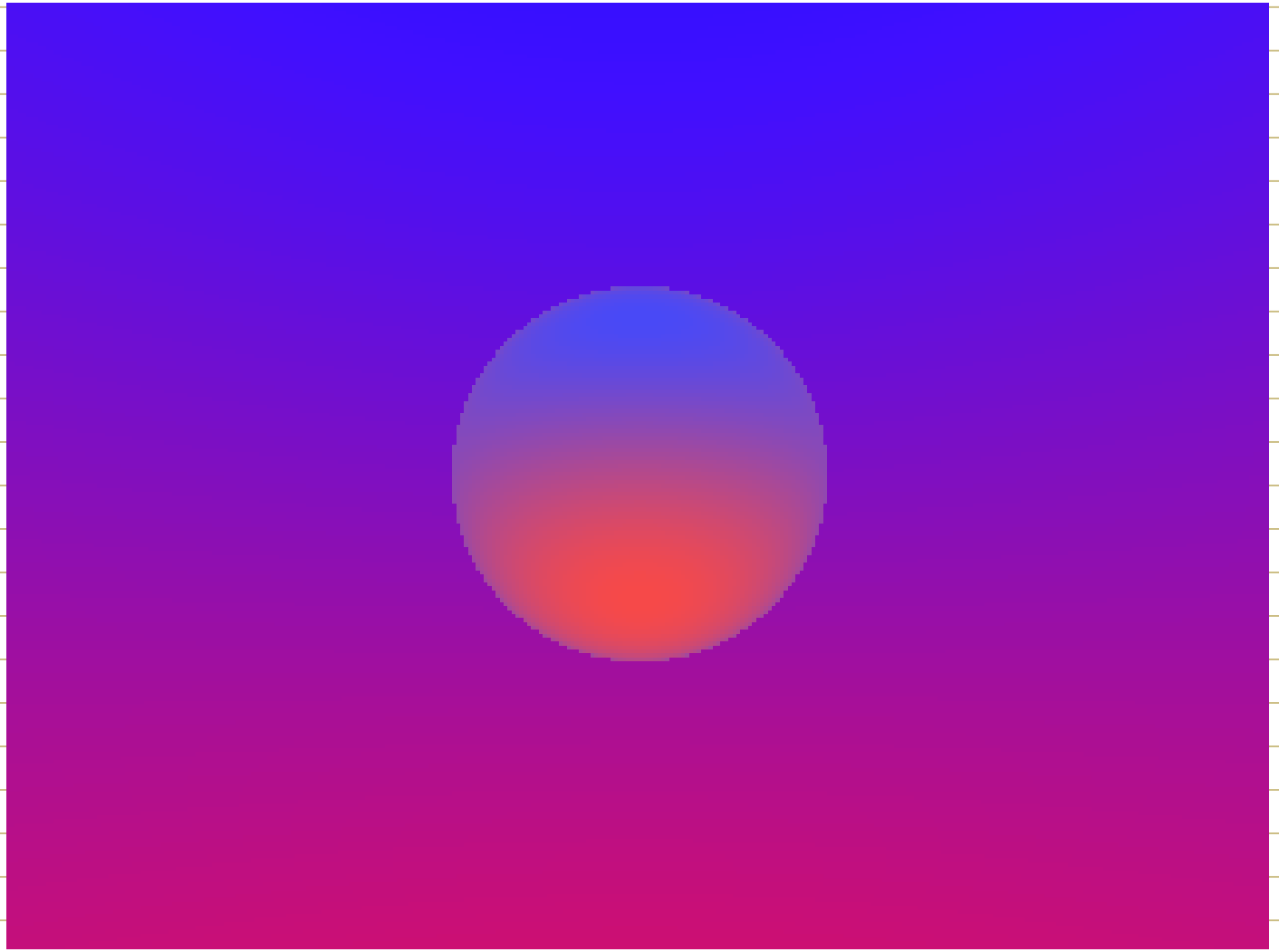
Słowo kluczowe sky_sphere może być użyte do łatwego tworzenia elementów nieba takich jak chmury, słońce, gwiazdy itp.

Płynne przejście kolorów nieba (parametr gradient) jest możliwe poprzez parametr gradient.

Przykład. Scena, w której zdefiniowano niebo

```
#include "colors.inc"
camera
{location <0, 1, -4>
look_at <0, 2, 0>
angle 80}
light_source { <10, 10, -10> White }
sphere
{2*y, 1
pigment { color rgb <1, 1, 1> }
finish { ambient 0.2 diffuse 0 reflection 0.6 }}
sky_sphere
{pigment {
gradient y
color_map {
[0 color Red]
[1 color Blue]
}
scale 2
translate -1
}}
```

Wynik

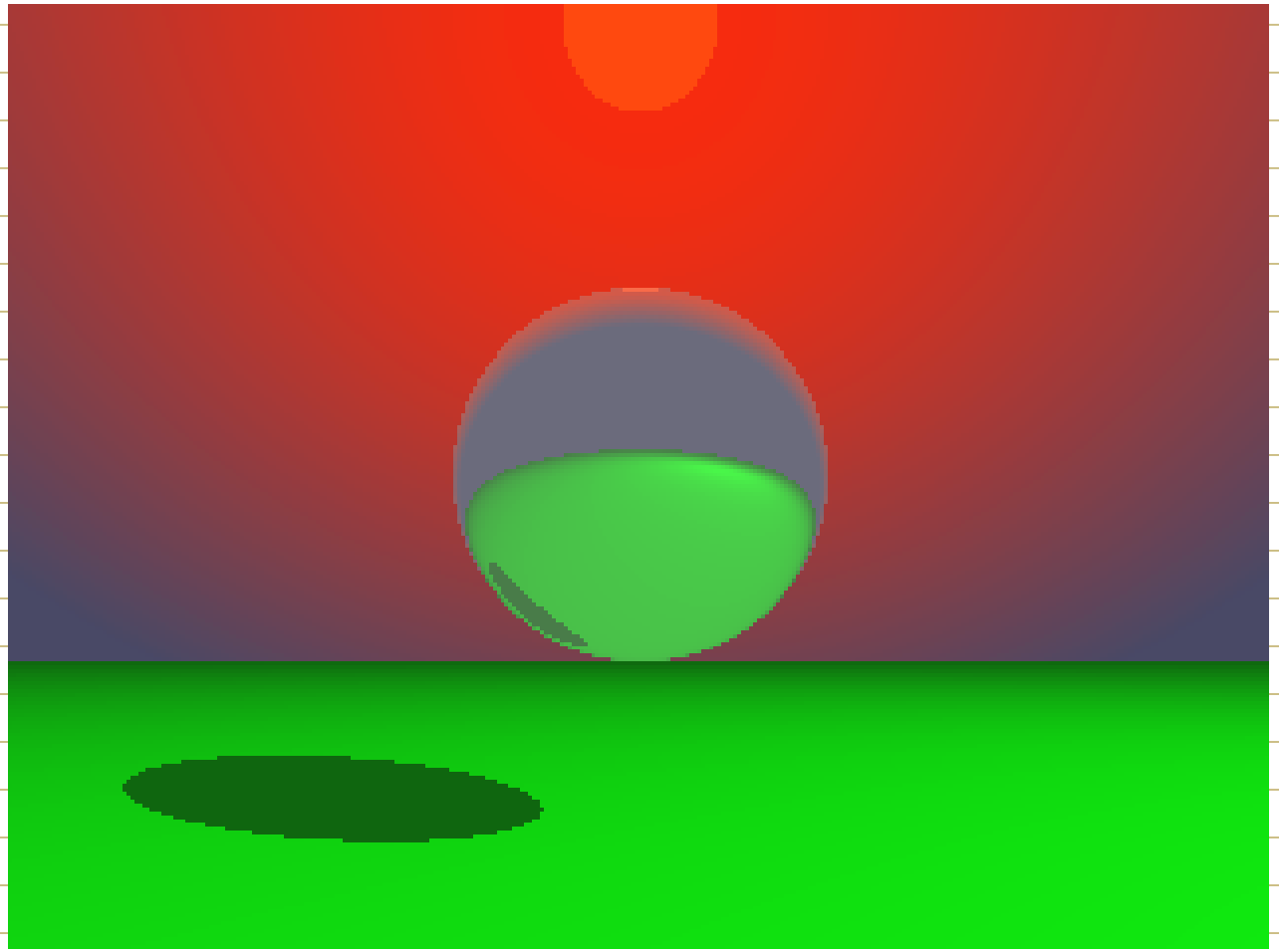


Dodawanie efektu słońca

W definicji obszaru nieba możliwe jest umieszczenia słońca, elementem kluczowym jest tutaj modyfikacja mapy kolorów.

```
sky_sphere
{
pigment {
gradient y
color_map {
[0.000 0.002 color rgb <1.0, 0.2, 0.0>
color rgb <1.0, 0.2, 0.0>]
[0.002 0.200 color rgb <0.8, 0.1, 0.0>
color rgb <0.2, 0.2, 0.3>]}
scale 2
translate -1
}
rotate -135*x}
```

Wynik



Dodawanie efektu chmur

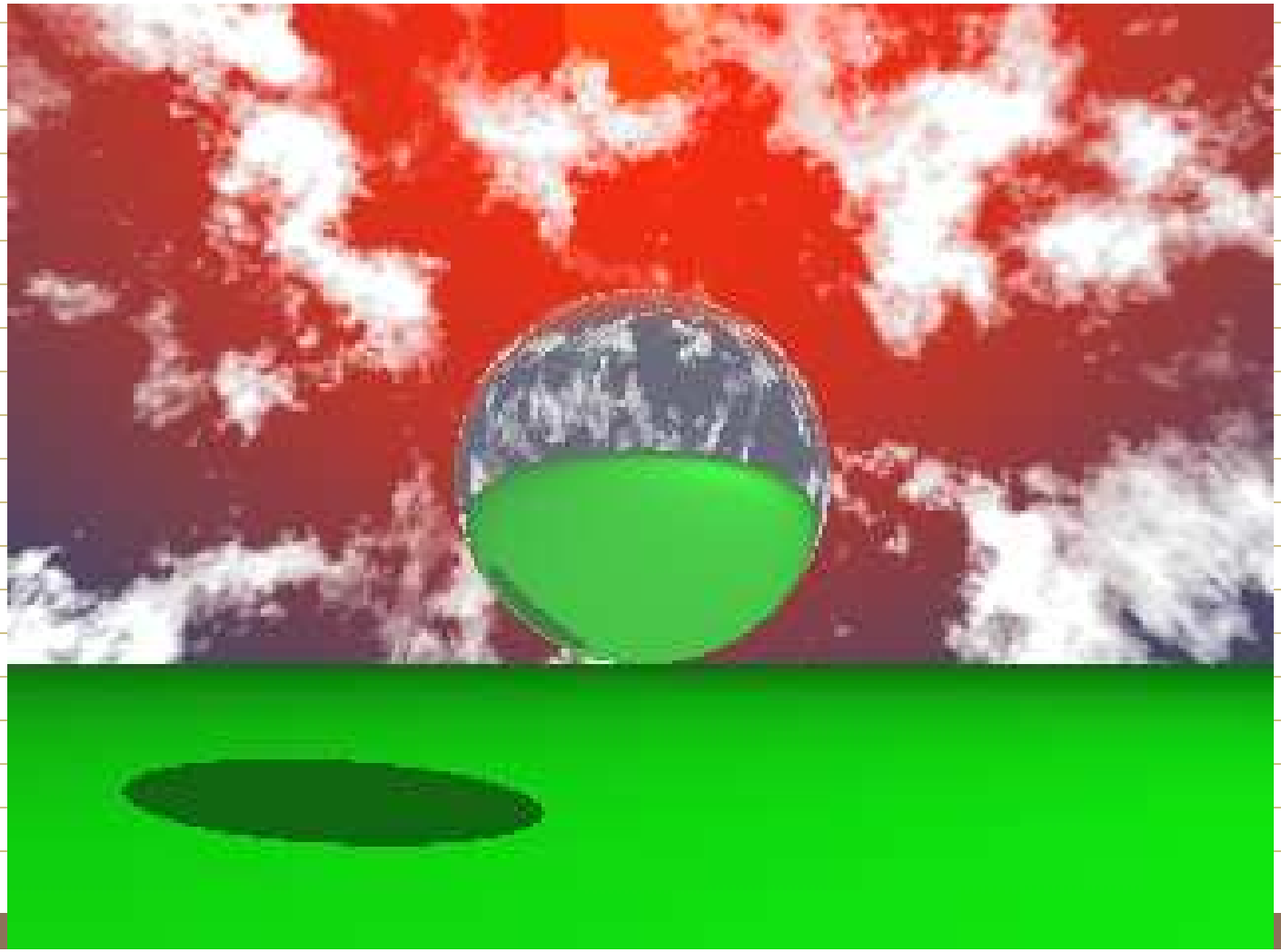
- W celu podniesienia efektywności obrazu nieba możliwe jest dodanie grup chmur zdefiniowanych jako drugi pigment. Do wytworzenia realistycznych chmur zalecane jest użycie wzorca bozo. Pigment ten, zadeklarowany jako drugi w kolejności w obszarze nieba (sky_sphere), będzie warstwą wierzchnią, która przykrywa tło nieba. Mapa kolorów powinna opisywać wiele odcieni kolorów.

Przykład. Definicja obszaru nieba z dodatkowym pigmentem opisującym chmury

```
sky_sphere
{pigment {
gradient y
color_map {
[0.000  0.002  color  rgb
  <1.0, 0.2, 0.0>
color  rgb  <1.0,  0.2,
  0.0>]
[0.002  0.200  color  rgb
  <0.8, 0.1, 0.0>
color  rgb  <0.2,  0.2,
  0.3>]}
scale 2
translate -1
}
```

```
pigment {bozo
turbulence 0.65
octaves 6
omega 0.7
lambda 2
color_map {
[0.0 0.1 color rgb <0.85, 0.85, 0.85>
color rgb <0.75, 0.75, 0.75>]
[0.1 0.5 color rgb <0.75, 0.75, 0.75>
color rgbt <1, 1, 1, 1>]
[0.5 1.0 color rgbt <1, 1, 1, 1>
color rgbt <1, 1, 1, 1>]
}
scale <0.2, 0.5, 0.2>
rotate -135*x}
```

Wynik



Dodawanie efektu mgły

Efekt mgły w języku POV-Ray'a można uzyskać w dwóch typach: mgła jednolita i mgła gruntowa.

Mgła jednolita ma jednakowy stopień gęstości na całym obszarze sceny, natomiast w przypadku mgły gruntowej gęstość zmniejsza się ku górze.

Mgła jednolita.

Postać ogólna:

```
fog
{
distance n
color rgb<r, g, b>
}
```

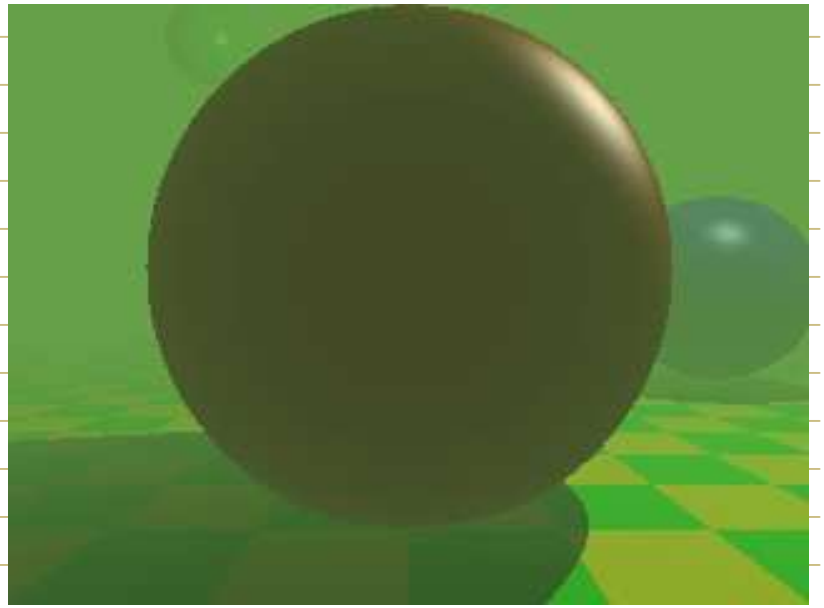
gdzie:

n – wartość parametru distance,

<x, y, z> – wektor określający kolor.

Mgła jednolita

Najprostszym typem mgły jest mgła jednolita, która ma jednakową gęstość w każdym punkcie. Parametry mgły ustalają słowa kluczowe distance i color. Atrybut distance określa zakres widoczności we mgle, natomiast parametr color barwę mgły. Stosując czarną mgłę można symulować ograniczenie zasięgu wizji.



Mgła gruntowa.

Postać ogólna:

```
fog
{
distance 150
color rgbf<r, g, b, f>
fog_type m
fog_offset n
fog_alt p
}
```

gdzie:

<r, g, b, f> – wektor określający kolor z uwzględnieniem przezroczystości,

m – typ mgły,

n – wartość Y, poniżej której następuje spadek mgły,

p – szybkość spadku gęstości mgły.

Wiele interesujących typów mgły oferuje mgła gruntowa. Obsługuje ona parametr fog_type, który określa typ mgły. Mgła gruntowa posiada ponadto parametry fog_offset i fog_alt.

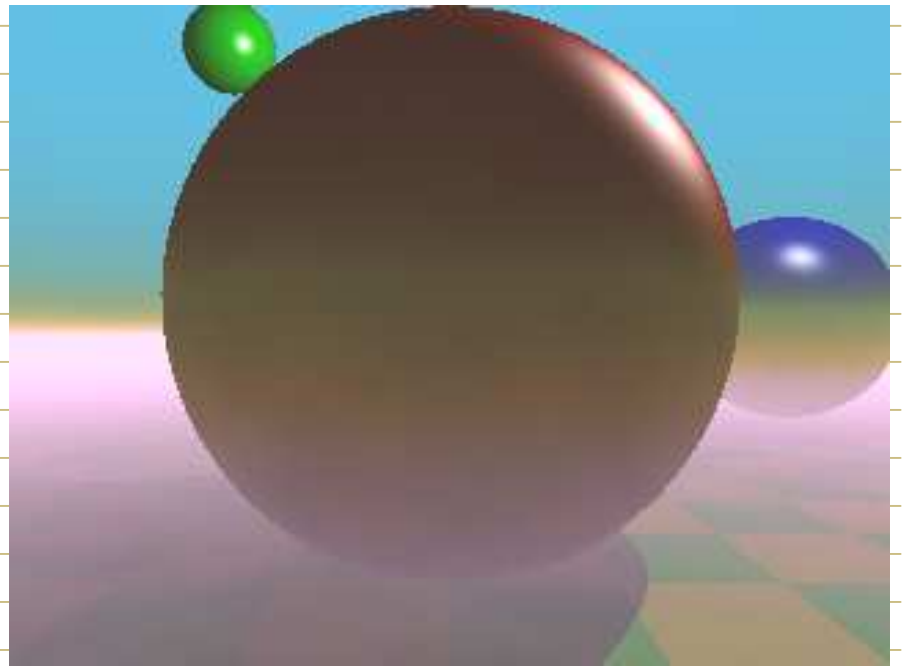
Parametr fog_offset określa wartość Y, poniżej której mgła zaczyna gęstnieć. Parametr fog_alt określa, jak szybko – wzdłuż osi Y – wartość mgły zbliża się do zera.

Przykład. Definicja mgły gruntowej

```
fog
{
distance 150
color  rgbf<0.3,  0.5,
   0.2, 1.0>
fog_type 2
fog_offset 25
fog_alt 1
}
```

Zastosowanie kilku powłok mgły

Język POV-Ray'a daje możliwość deklarowania kilku powłok mgły w jednej scenie. Mgła rozmieszczona w różnych miejscach sceny czyni ją bardziej naturalną, gdyż – podobnie jak w świecie rzeczywistym – ma niejednolitą strukturę.



Efekt tęczy

Postać ogólna:

```
Rainbow
{
angle n
width m
direction <x, y, z>
jitter
color_map
{
[kolor1]
[kolor2]
[kolor3]
        }
}
```

gdzie:

n – nachylenie tęczy,

m – szerokość tęczy,

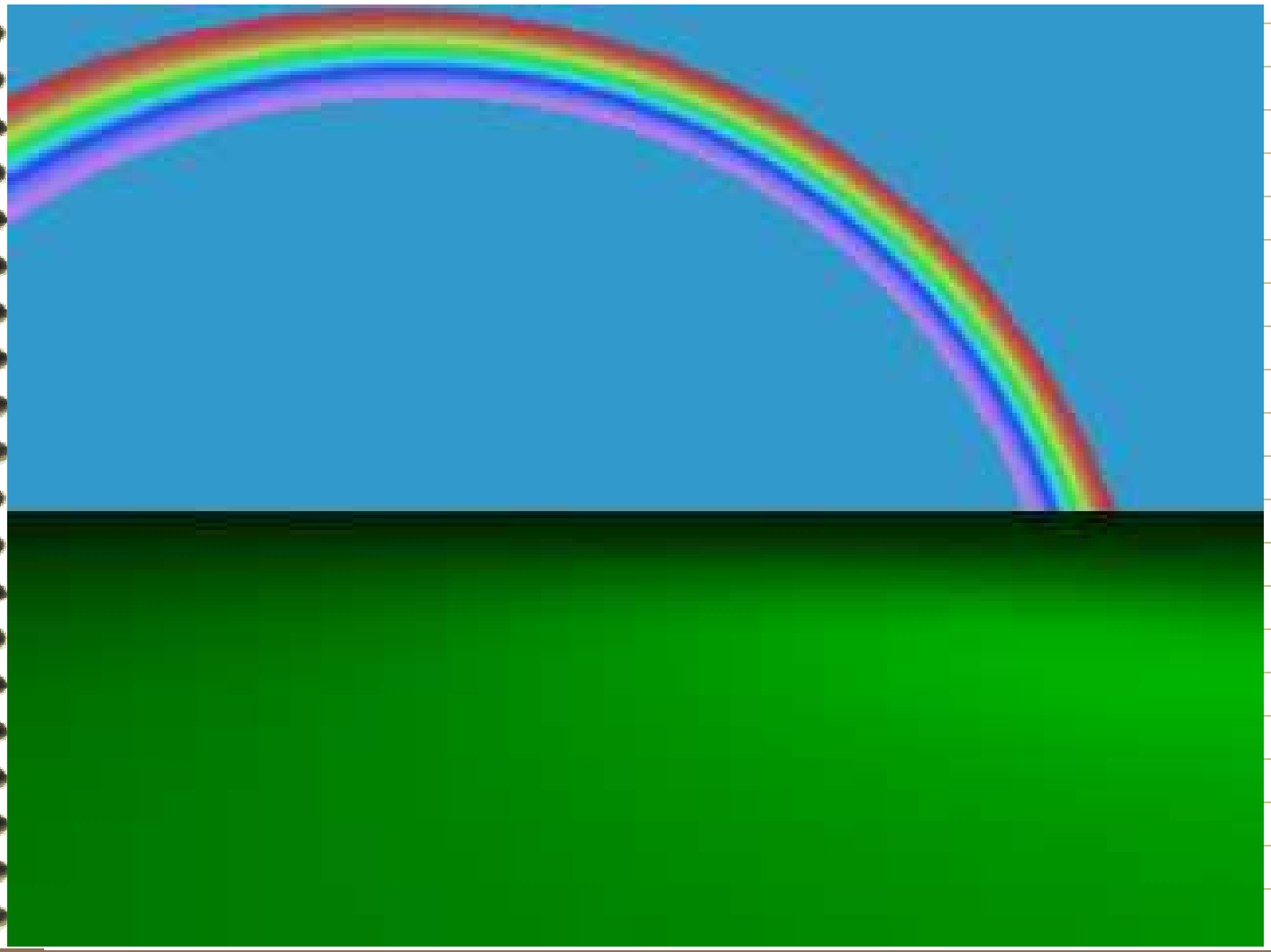
<x, y, z> – wektor skalujący tęczę.

Wielkość i kształt tego zjawiska opisują słowa kluczowe angle i width. Częścią kluczową opisu tęczy jest deklaracja kolorów, które ją tworzą. Dopiero tak zdefiniowane kolory mogą być użyte do definicji tęczy.

Przykład. Zjawisko tęczy

```
#include "colors.inc"
camera
{location <0, 20, -100>
look_at <0, 25, 0>
angle 80}
background {color SkyBlue}
plane { y, -10 pigment { color Green } }
light_source {<100, 120, 40> color White}
#declare r_violet1 = color rgbf<1.0, 0.5, 1.0, 1.0>;
#declare r_violet2 = color rgbf<1.0, 0.5, 1.0, 0.8>;
#declare r_indigo = color rgbf<0.5, 0.5, 1.0, 0.8>;
#declare r_blue = color rgbf<0.2, 0.2, 1.0, 0.8>;
#declare r_cyan = color rgbf<0.2, 1.0, 1.0, 0.8>;
#declare r_green = color rgbf<0.2, 1.0, 0.2, 0.8>;
#declare r_yellow = color rgbf<1.0, 1.0, 0.2, 0.8>;
#declare r_orange = color rgbf<1.0, 0.5, 0.2, 0.8>;
#declare r_red1 = color rgbf<1.0, 0.2, 0.2, 0.8>;
#declare r_red2 = color rgbf<1.0, 0.2, 0.2, 1.0>;
rainbow
{angle 42.5
width 5
distance 1.0e7
direction <-0.2, -0.2, 1>
jitter 0.01
color_map {
[0.000 color r_violet1]
[0.100 color r_violet2]
[0.214 color r_indigo]
[0.328 color r_blue]
[0.442 color r_cyan]
[0.556 color r_green]
[0.670 color r_yellow]
[0.784 color r_orange]
[0.900 color r_red1]}}
```

Wynik



Transformacje (przekształcenia) obiektów

Translacja

Postać ogólna:

translate <x, y, z>

gdzie:

<x, y, z> – wektor, o który przesuwamy obiekt.

Słowo kluczowe translate umożliwia przesuniecie obiektu względem jego pierwotnego położenia o podany wektor standardowy <x, y, z>

Polecenie translate obiektu należy umieścić wewnątrz definicji obiektu.

Poniżej został podany przykład programu, w którym wykorzystano translację.

Przykład. Przesunięcie pierścienia o wektor <0, 2, 0>

```
torus
{
3, 11
pigment { Red }
translantate <0, 2, 0>
}
```

Rotacja

Postać ogólna:

rotate <x, y, z>

gdzie:

<x, y, z> – wektor, o który obracamy obiekt. Składowe x, y, z są kątami w stopniach.

Rotacja (inaczej ruch obrotowy) umożliwia obrót obiektu względem osi układu współrzędnych. Obrót obiektu określany jest na podstawie wektora <x, y, z> .w którym wartości liczbowe odpowiadają obrotowi danego obiektu względem osi w stopniach.

Poniżej został podany przykład programu, w którym wykorzystano rotację.

Przykład. Rotacja (obrót) pierścienia o -45° wokół osi X

```
torus
{
3, 11
pigment { Red }
rotate <-45, 0, 0>
}
```

Skalowanie

Postać ogólna:

scale <x, y, z>

gdzie:

<x, y, z> – wektor, o który obracamy obiekt. Składowe x, y, z określają skalę powiększenia.

Skalowanie zmienia rozmiar obiektu względem jego wymiarów pierwotnych. Skalowanie opisujemy za pomocą wektora <x, y, z>. Elementy wektora określają skalę powiększenia (lub pomniejszenia) względem danej osi. Wartość 1.0 pozostawia wymiar obiektu bez zmian, wartość 0.0 obiekt pomniejszony, zaś 2.0 – powiększony.

Przykład. Scena, w której zastosowano transformacje obiektu

```
#include "colors.inc"
#include "textures.inc"
camera
{location <-6, 2, -9>
look_at <-2, 2.5, 0>
angle 48}
plane
{ y, -1
texture {
pigment {checker color rgb<0.5, 0, 0>  color rgb<0, 0.5, 0.5> }
finish {
diffuse 0.4
ambient 0.2
phong 1
phong_size 100
reflection 0.25
}}}
torus
{1.5, 0.5
texture { Brown_Agate }
rotate <90, 180, 0>
translate <-1, 1, 3>
scale <1, 2, 1>}
light_source {<2, 10, -3> color White}
```

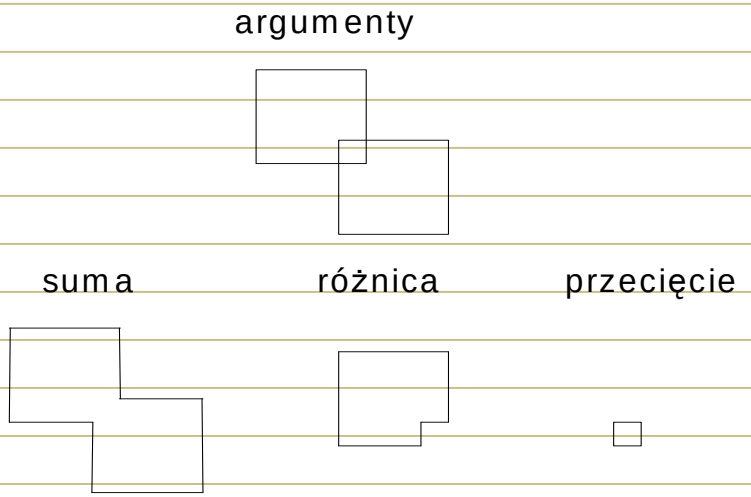
Wynik

Operacje na obiektach – technika CSG

- Konstruktywna Teoria Brył (ang. Constructive Solid Geometry – CSG) jest efektywną metodą budowania brył geometrycznych o skomplikowanych kształtach przy pomocy operacji mnogościowych wykonywanych na prostych bryłach. Obiekty utworzone na bazie kształtów podstawowych nie zawsze odpowiadają oczekiwanym wynikom. Technika CSG pozwala modelować pojedyncze elementy (detale) obiektu a następnie połączyć je w jedną całość, tworzącą nowy obiekt.

Konstruktywna teoria brył

- Metoda CSG obejmuje takie operacje jak łączenie (ang. union lub merge), część wspólna (ang. intersection) i odejmowanie obiektów (ang. difference). Na rys. została przedstawiona filozofia podstawowych operacji CSG, na przykładzie dwóch kwadratów.



Operacja "union"

Postać ogólna:

union {

{definicja_obiektu1}

{definicja_obiektu2}

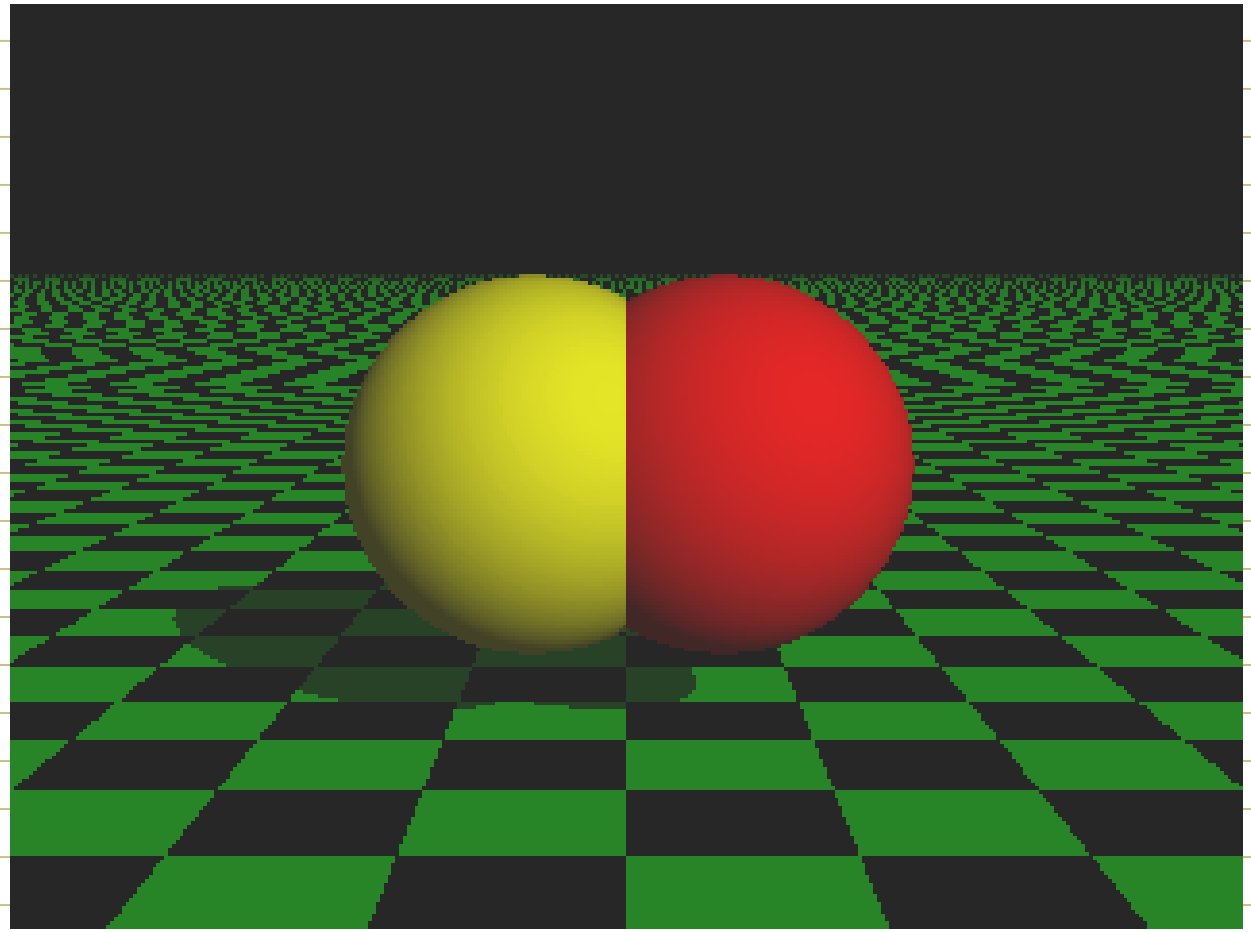
}

Operacja union polega na łączeniu grupy obiektów w jeden obiekt. Wynik tej operacji traktowany jest jako nowy obiekt w scenie.

Przykład

```
#include "colors.inc"
camera
{location <0, 1, -10>
look_at 0
angle 36}
light_source { <500, 500, -1000> White }
plane
{  y, -1.5
pigment { checker Green Black }}
union
{sphere
{ <0, 0, 0>, 1
translate -0.5*x}
sphere
{ <0, 0, 0>, 1
translate 0.5*x}
pigment {Blue}
scale <1, 25, 1>
rotate <30, 0, 45>}
```

Wynik



Operacja "intersection"

Postać ogólna:

intersection

{

{definicja_obiektu1}

{definicja_obiektu2}

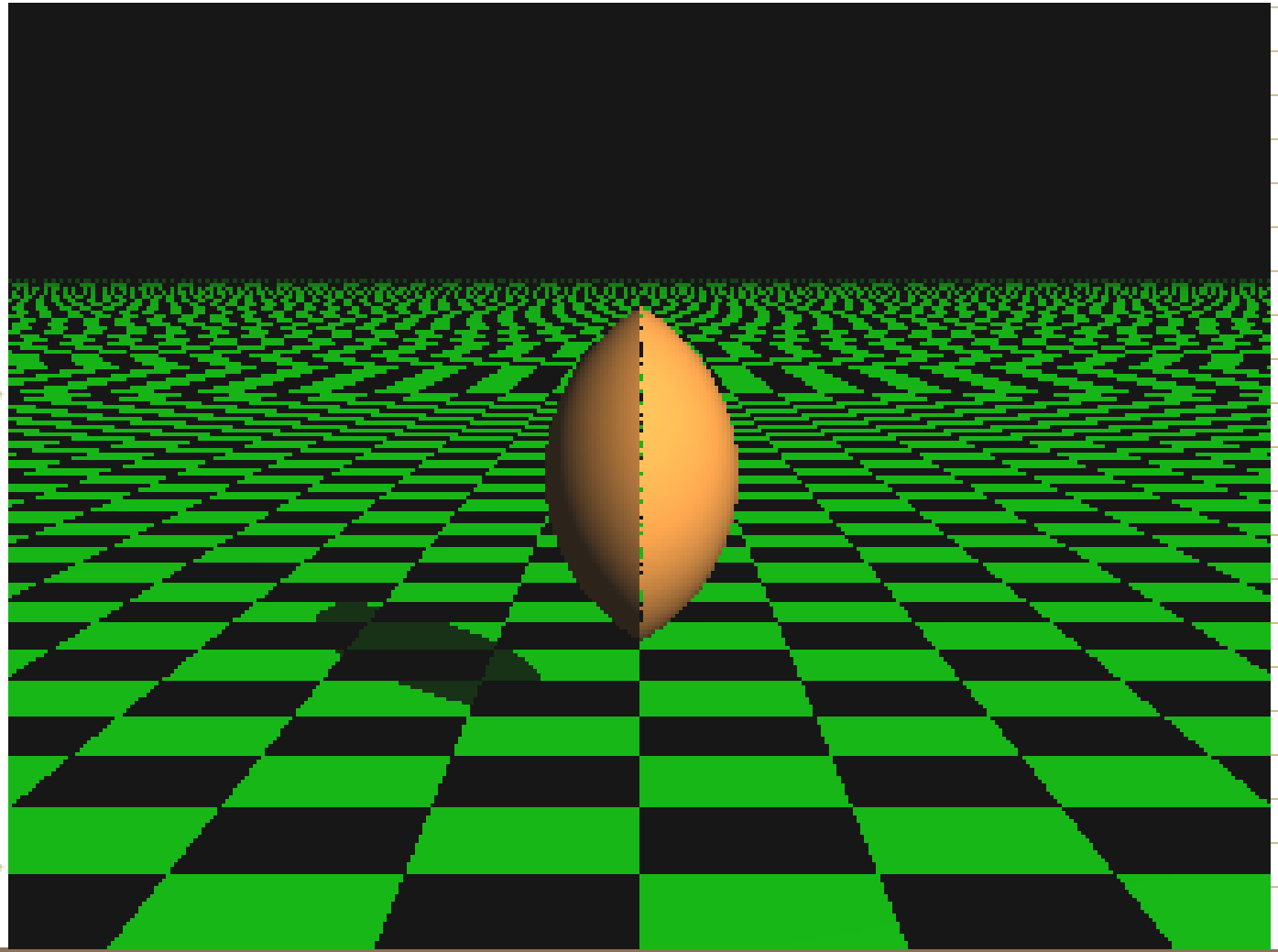
}

Operacja intersection polega na tworzeniu obiektów będących częścią wspólną obiektów składowych. Część zewnętrzna obiektów zostaje usunięta, a wygenerowany zostaje tylko wspólny obszar dla wszystkich obiektów. Obszar ten staje się nowym obiektem o jednorodnej strukturze.

Przykład

```
#include "colors.inc"
camera
{location <0, 1, -10>
look_at 0
angle 36}
light_source { <500, 500, -1000> White }
plane
{y, -1.5
pigment { checker Green Black }}
intersection
{sphere
{<0, 0, 0>, 1
translate -0.5*x}
sphere
{<0, 0, 0>, 1
translate 0.5*x}
pigment {Gold}}
```

Wynik



Operacja "difference"

Postać ogólna:

difference

{

{definicja_obiektu1}

{definicja_obiektu2}

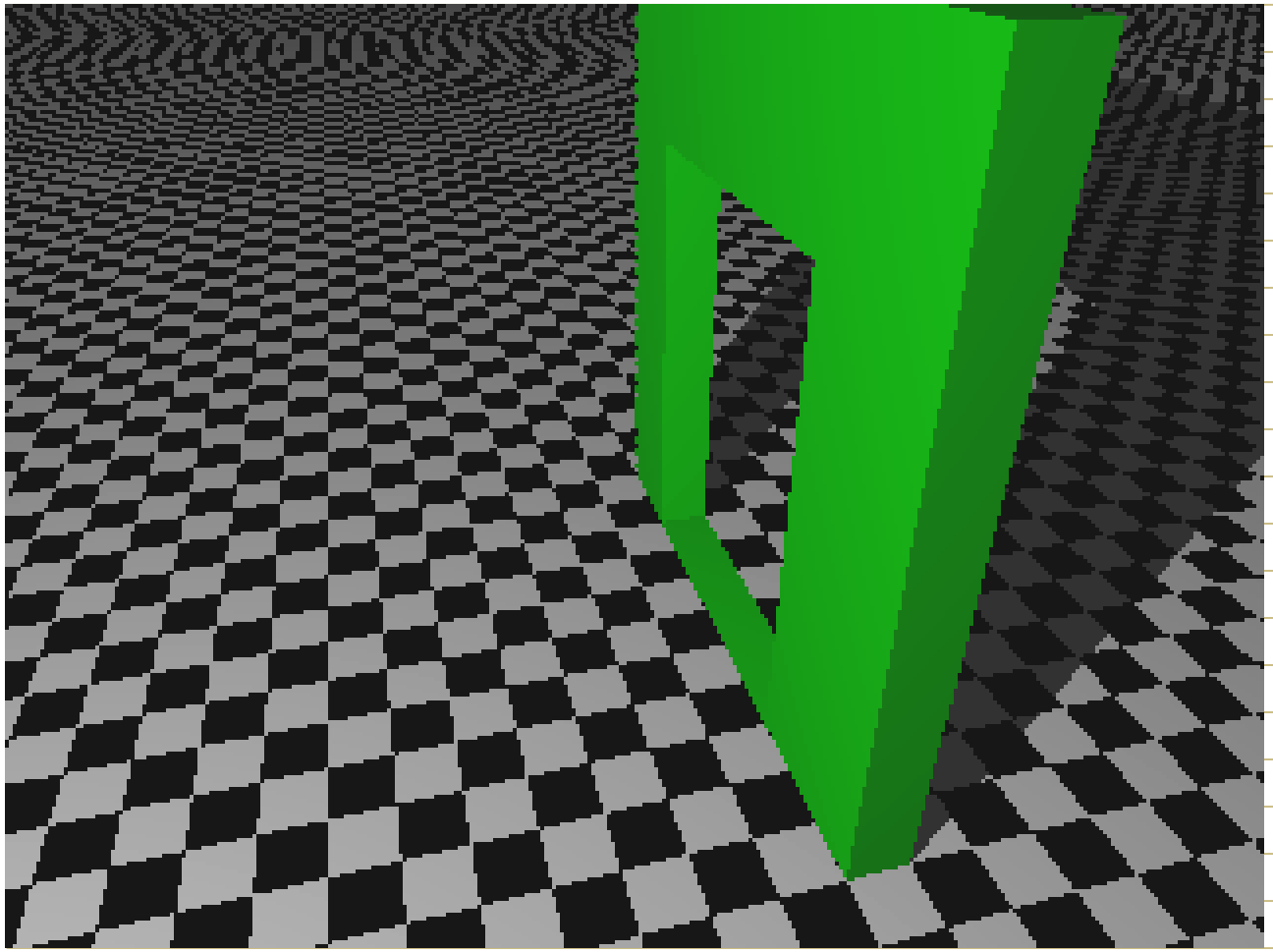
}

Operacja difference polega na odejmowaniu obiektów. Działanie to wymaga zapisania definicji obiektów w odpowiedniej kolejności: najpierw obiekt główny od którego odejmujemy, a następnie obiekty, które są odejmowane od obiektu głównego.

Przykład

```
#include "colors.inc"
camera
{
location <18, 12,-5>
look_at <0,0,0>
}
light_source {<20,15,-15> White}
plane
{
<0,1,0>,0
pigment {checker color <0,0,0> color <1,1,1> }}
difference
{
box { <0,0,0>, <11,11,1> }
box { <2,0,-1>, <8,8,2> }
pigment { Green } }
```

Wynik



Operacja "merge"

Postać ogólna:

merge

{

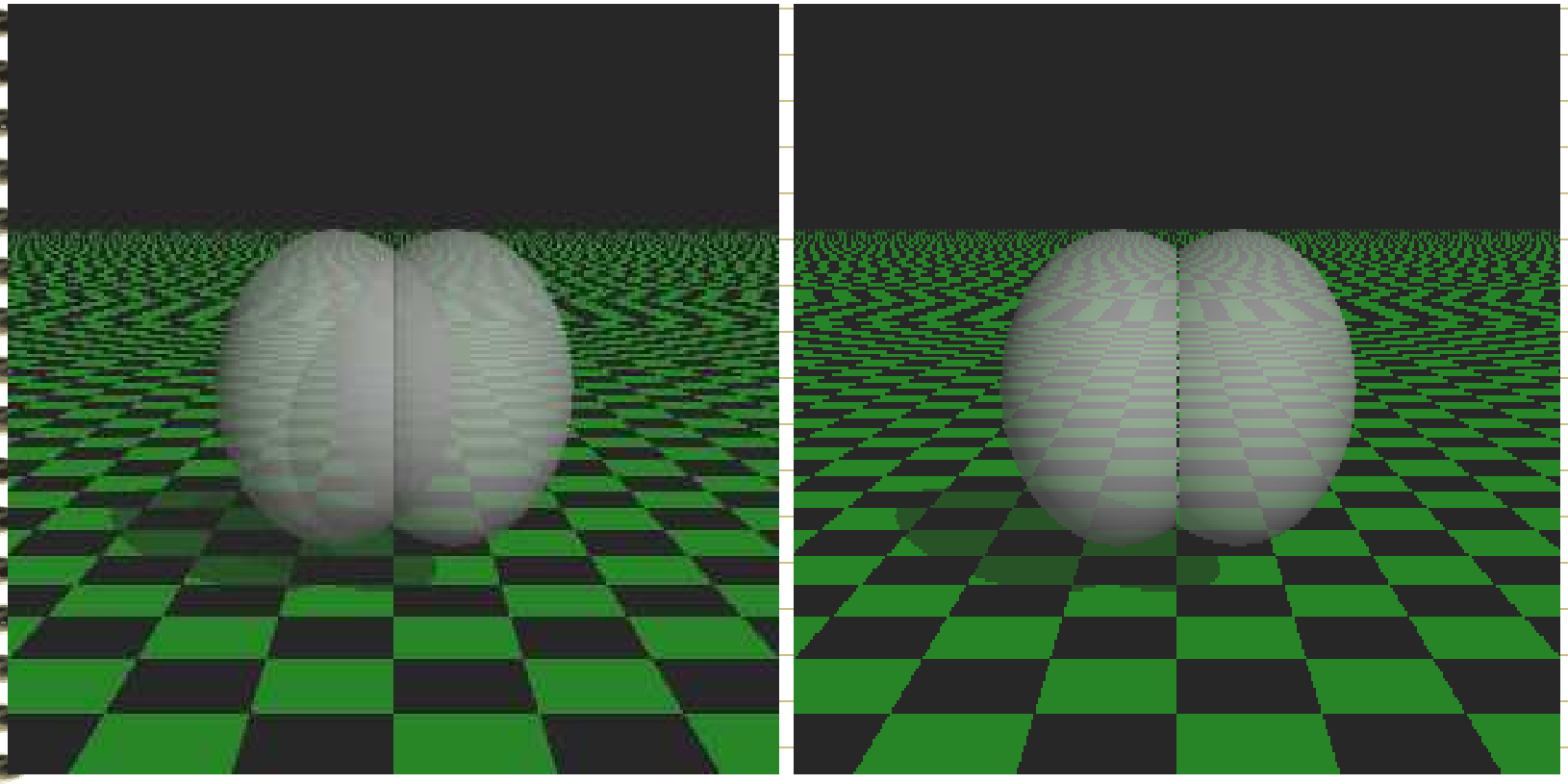
{obiekt1}

{obiekt2}

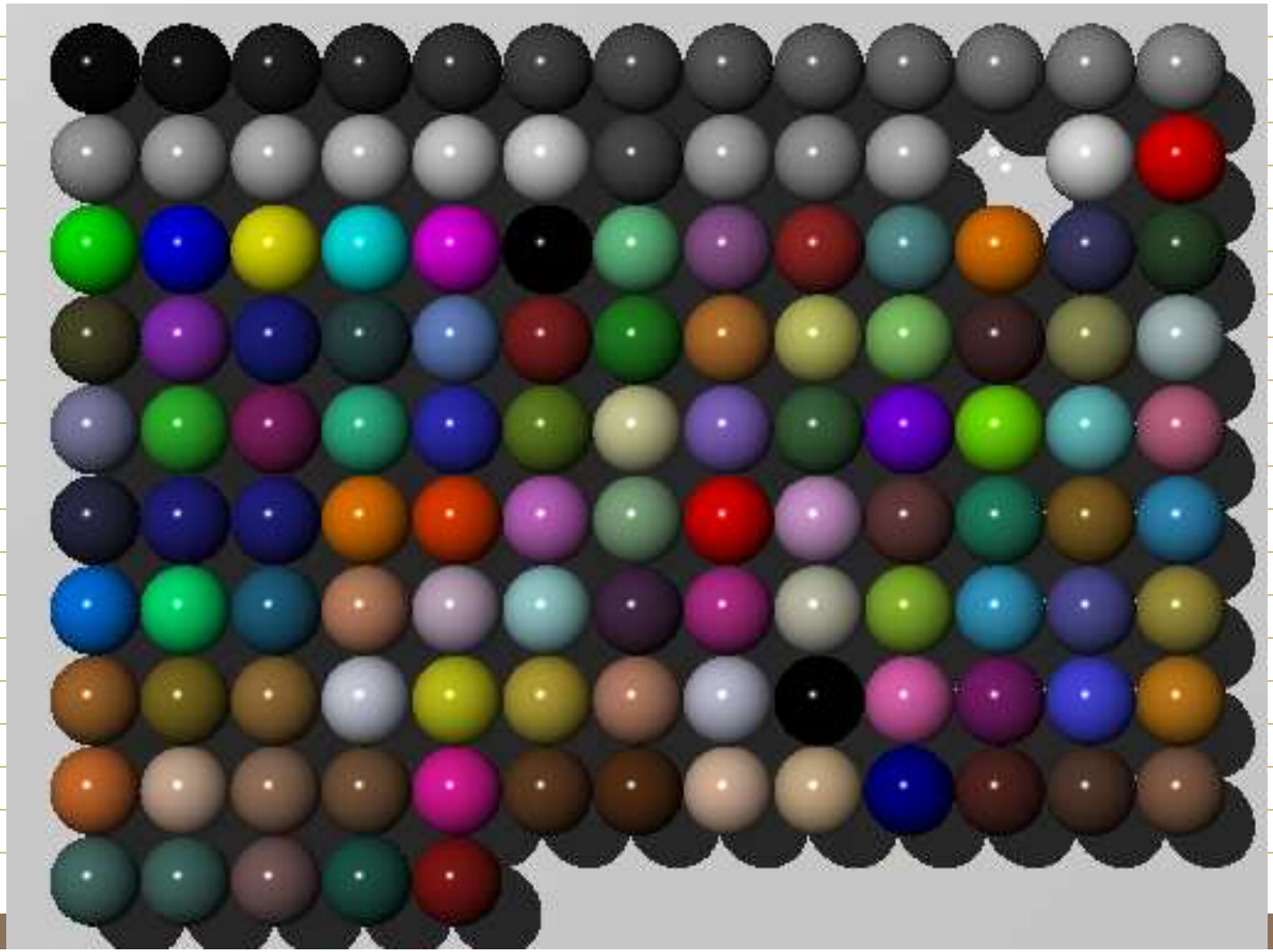
}

Operacja merge jest bardzo podobna do operacji union. Zasadnicza różnica polega na tym, że po zastosowaniu operacji merge nie jest poddawane operacji śledzenia wnętrze obiektów. W przypadku łączenia przezroczystych obiektów zachodzących na siebie użycie operacji merge umożliwia uzyskanie ich jednolitej struktury bez widocznych wewnętrznych krawędzi.

Przykład dla union i merge



Plik colors.inc



Nazwy kolorów w colors.inc

kolumna	1	wiersz	1	pigment {Gray05}
kolumna	2	wiersz	1	pigment {Gray10}
kolumna	3	wiersz	1	pigment {Gray15}
kolumna	4	wiersz	1	pigment {Gray20}
kolumna	5	wiersz	1	pigment {Gray25}
kolumna	6	wiersz	1	pigment {Gray30}
kolumna	7	wiersz	1	pigment {Gray35}
kolumna	8	wiersz	1	pigment {Gray40}
kolumna	9	wiersz	1	pigment {Gray45}
kolumna	10	wiersz	1	pigment {Gray50}
kolumna	11	wiersz	1	pigment {Gray55}
kolumna	12	wiersz	1	pigment {Gray60}
kolumna	13	wiersz	1	pigment {Gray65}
kolumna	1	wiersz	2	pigment {Gray70}

Plik textures.inc zestaw 1

Nazwy textur w textures.inc zestaw 1

kolumna	1	wiersz	1	texture { pigment { Jade } },
kolumna	2	wiersz	1	texture { pigment { Red_Marble } },
kolumna	3	wiersz	1	texture { pigment { White_Marble } },
kolumna	4	wiersz	1	texture { pigment { Blood_Marble } },
kolumna	5	wiersz	1	texture { pigment { Blue_Agate } },
kolumna	6	wiersz	1	texture { pigment { Sapphire_Agate } },
kolumn				

Plik textures.inc zestaw 2

Nazwy textur w textures.inc zestaw 2

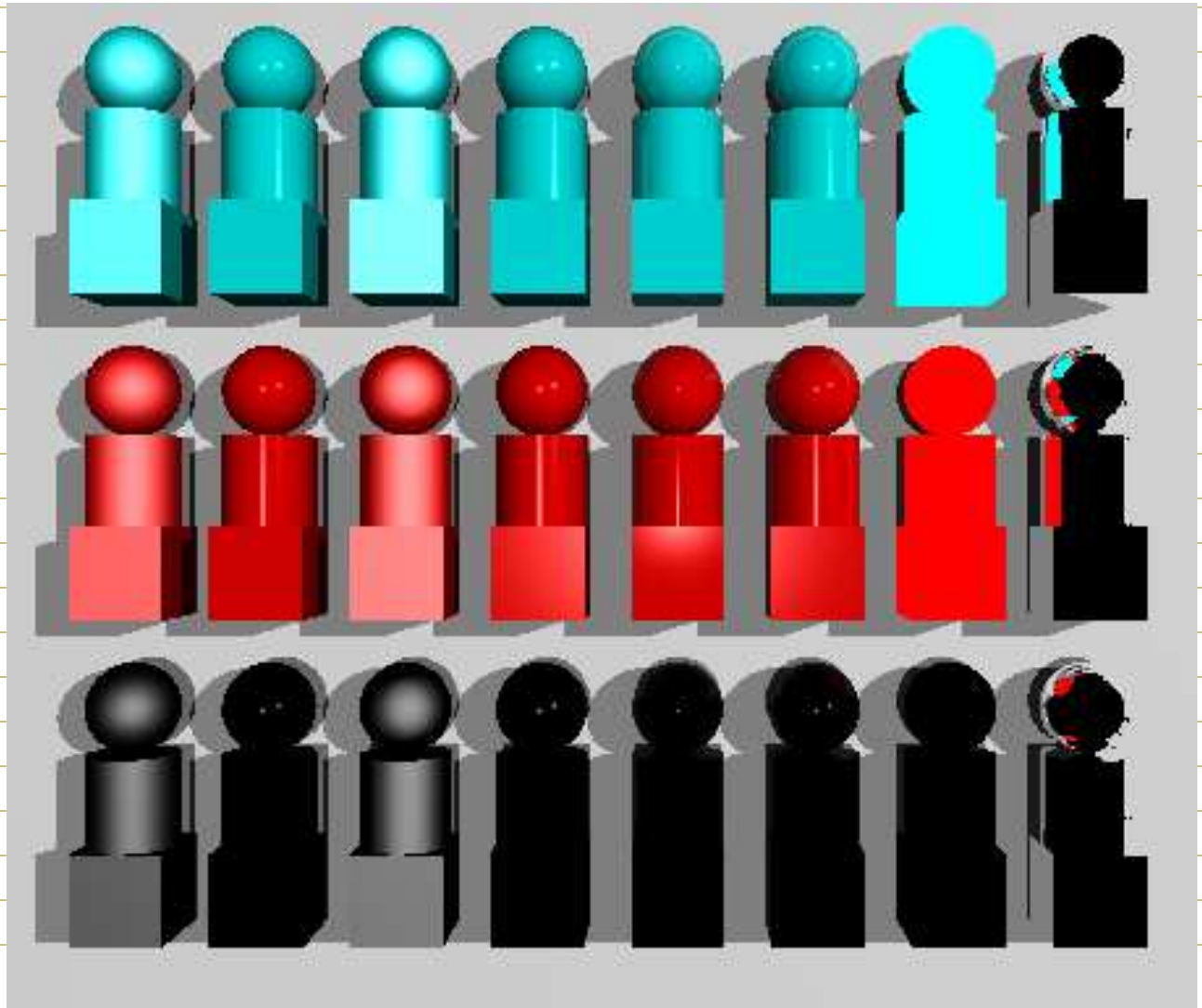
kolumna	1	wiersz	1	texture { pigment { DMFWood4 } },
kolumna	2	wiersz	1	texture { pigment { DMFWood5 } },
kolumna	3	wiersz	1	texture { DMFWood6 },
kolumna	4	wiersz	1	texture { pigment { DMFLightOak } },
kolumna	5	wiersz	1	texture { pigment { DMFDarkOak } },
kolumna	6	wiersz	1	texture { EMBWood1 },

Plik textures.inc zestaw 3

Nazwy textur w textures.inc zestaw 3

kolumna	1	wiersz	1	texture {	Copper_Metal },
kolumna	2	wiersz	1	texture {	Polished_Chrome },
kolumna	3	wiersz	1	texture {	Polished_Brass },
kolumna	4	wiersz	1	texture {	New_Brass },
kolumna	5	wiersz	1	texture {	Spun_Brass },
kolumna	6	wiersz	1	texture {	Brushed_Aluminum },

Plik "finish.inc"



Nazwy „finish

kolumna	1	wiersz	1	texture { pigment {rgb <0,1,1>} finish {Dull } },
kolumna	2	wiersz	1	texture { pigment {rgb <0,1,1>} finish {Shiny } },
kolumna	3	wiersz	1	texture { pigment {rgb <0,1,1>} finish {Phong_Dull } },
kolumna	4	wiersz	1	texture { pigment {rgb <0,1,1>} finish {Phong_Shiny } },
kolumna	5	wiersz	1	texture { pigment {rgb <0,1,1>} finish {Glossy } },
kolumna	6	wiersz	1	texture { pigment {rgb <0,1,1>} finish {Phong_Glossy } },
kolumna	7	wiersz	1	texture { pigment {rgb <0,1,1>} finish {Luminous } },

Animacja

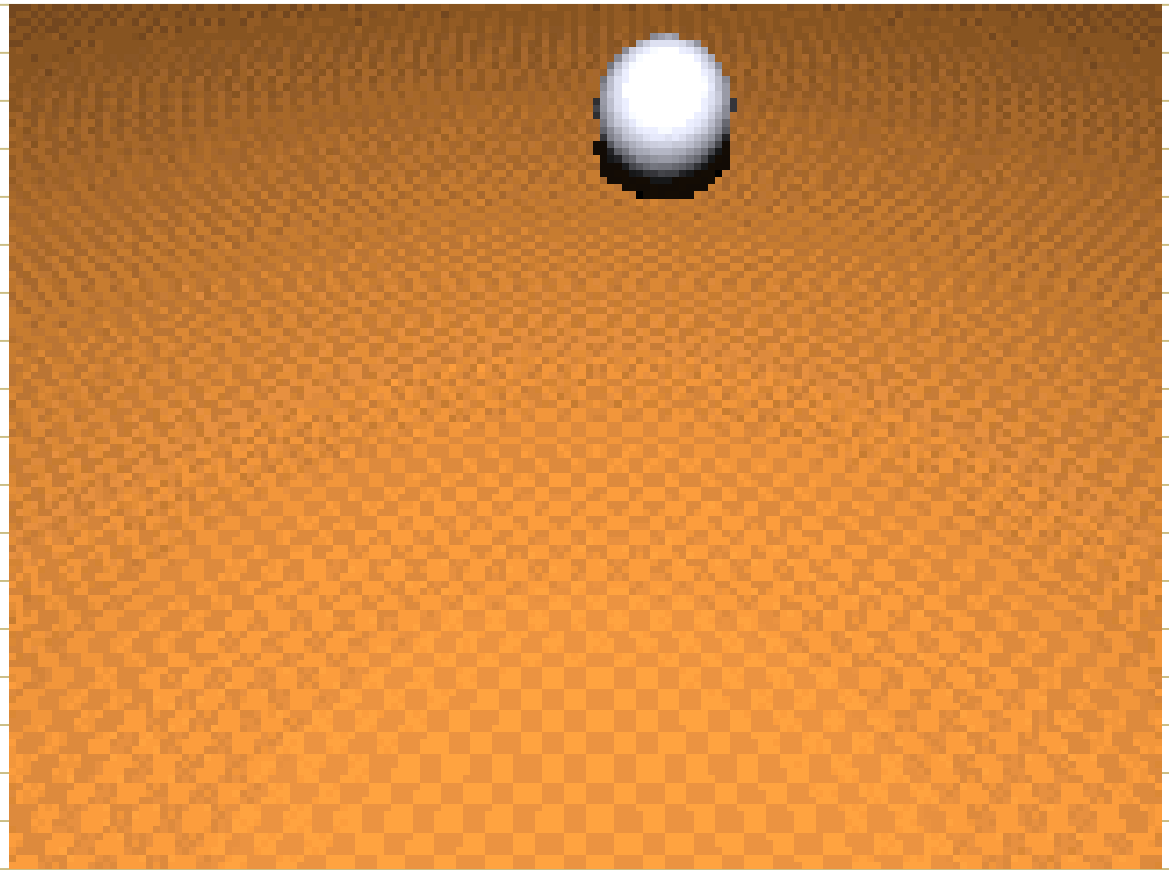
Aby wygenerować animację sceny 3-D należy:

- zaplanować przebieg animacji i opisać matematycznie trajektorie, po których mają się przemieszczać ruchome obiekty sceny,
- opisać scenę z uwzględnieniem ruchu obiektów,
- wygenerować ciąg klatek (obrazów),
- skompresować wytworzony ciąg klatek do postaci filmu, zapisanego w jednym z przyjętych formatów (MPEG, AVI, QuickTime, GIF animowany).

Krok 1.1

1. **Zaplanowanie przebiegu animacji i opisanie ruchu obiektu: Przykładowe zadanie jest następujące:**
   - **ponad płaszczyzną na wysokości h , po trajektorii będącej okręgiem o promieniu R, porusza się sfera o promieniu r,**
   - **środek okręgu trajektorii ruchu sfery znajduje się w punkcie (0, h, 0),**
   - **liczba klatek animacji wynosi N = 25**

Przykład



Krok 1.2

2. Sporządzenie opisu sceny w języku programu POV-ray Opis sceny dla wykonania animacji składa się z dwóch plików, pliku inicjalizacyjnego *.ini i pliku opisu sceny *.pov.

- **plik inicjalizacyjny *.ini zawiera informacje dotyczące parametrów procesu generacji ciągu klatek,**
- **w pliku *.pov znajduje się opis sceny składający się z opisu jej geometrii i algorytmu sterowania procesem generacji klatek .**

Przykład

```
;*****************************************************
; Przykładowy plik inicjalizacyjny dla animacji.
;*****************************************************
Input_File_Name = Orbita.pov    ; Nazwa pliku z opisem sceny
Output_File_Type = C                    ; Format pliku wynikowego (klatki)
                                ; C ozancza nieskompresowany format TGA

Output_File_Name= Orbita                ; Nazwa pliku (plików) wynikowych
Initial_Frame = 1               ; Numer pierwszej klatki animacji
Final_Frame = 25                ; Numer ostatniej klatki animacji
Initial_Clock = 1               ; Początkowa wartość zegara animacji
Final_Clock = 25                ; Końcowa wartość zegara aniamacji
```

Krok 1.3

3. Otwarcie pliku inicjalizacyjnego i wykonaniu polecenia Render > Start Rendering(lub Run) spowoduje, że

- **wykonanie programu *Input_File_Name* (Orbira.pov) nastapi tyle razy, ile wynosi liczba Final_Frame - Initial_Frame +1, w tym przypadku 25 razy,**
- **każde wykonanie programu spowoduje wygenerowanie obrazu (klatki) o nazwie Output_File_Name z kolejnym numerem i w formacie Output_File_Type, w powyższym przykładzie Orbita1.tga, Orbita2.tga, ..., Orbita25.tga,**
- **w czasie generowania kolejnych klatek zmienna wewnętrzna o nazwie clock (używana w opisie sceny Input_File_Name) zwiększa się liniowo od wartości Initial_Clock do Final_Clock, w tym przykładzie dla klatki pierwszej clock =1. drugiej clock =2 itd.**

Przykład programu orbita.pov

```
#include "colors.inc"
camera
{ location <0, 30, -40> look_at <0, 0, -10> }
 plane { <0, 1, 0>, -5 pigment {checker color Gold, color Copper } }
#declare Number = 25; // Liczba równa liczbie klatek aniamcji
#declare Ang = 360/Number;
sphere
{ <0, 0, 2>, 5 texture { pigment { color Silver } }
translate <0, 2, 20>
rotate <0, clock*Ang, 0>}
light_source { <0, 50, -30> color White}
light_source { <0, 50, -30> color White
```

Krok 2.**Przebieg procesu wykonywania animacji**

1. **Utworzyć folder roboczy.**
2. **Skopiować do utworzonego folderu pliki Orbita.ini, Orbita.pov, Animate.EXE (program do kompresji klatek) , Animate.TXT (instrukcja dla programu Animate)**
3. **Uruchomić (jeśli nie nastąpiło to wcześniej) program POV-ray.**
4. **Otworzyć pliki *Orbita.ini* i *Orbit.pov***
5. **Wybrać zakładkę z nazwą pliku Orbita.ini (treść pliku inicjalizacyjnego powinna być widoczna w oknie edytora programu POV-ray).**
6. **Ustawić wielkość generowanego obrazu (raczej najmniejszą, 160x120 No AA).**
7. **Nacisnąć przycisk Run i poczekać na wygenerowanie 25 klatek animacji.**
8. **Obejrzeć zawartość folderu roboczego.**
9. **Uruchomić (przy pomocy polecenia Uruchom lub Wiersz polecenia) program Animate, który dokona kompresji przygotowanego wcześniej ciągu klatek do formatu GIF- animowany, program należy uruchomić przez wpisanie w linii komendy Animate \v *.tga.**
10. **Obejrzeć wynik zawarty w pliku orbita.gif**