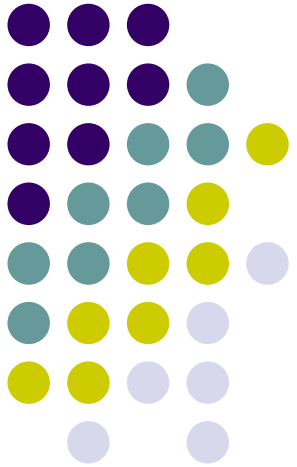
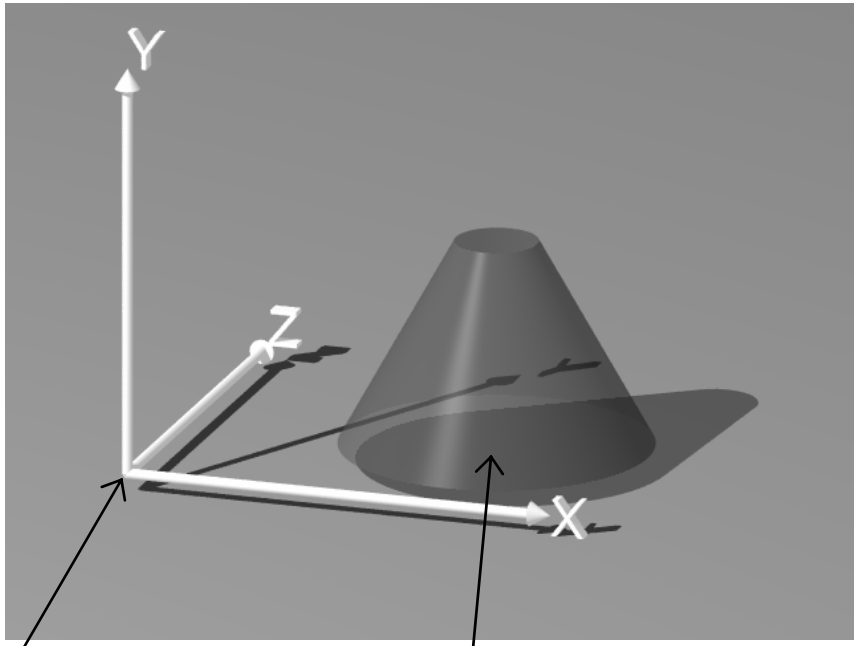
Математичні Основи

Координатні системи, Вектори



Координатні системи

- В КС розташування об'єктів відбувається зазвичай в загально прийнятій системі координат

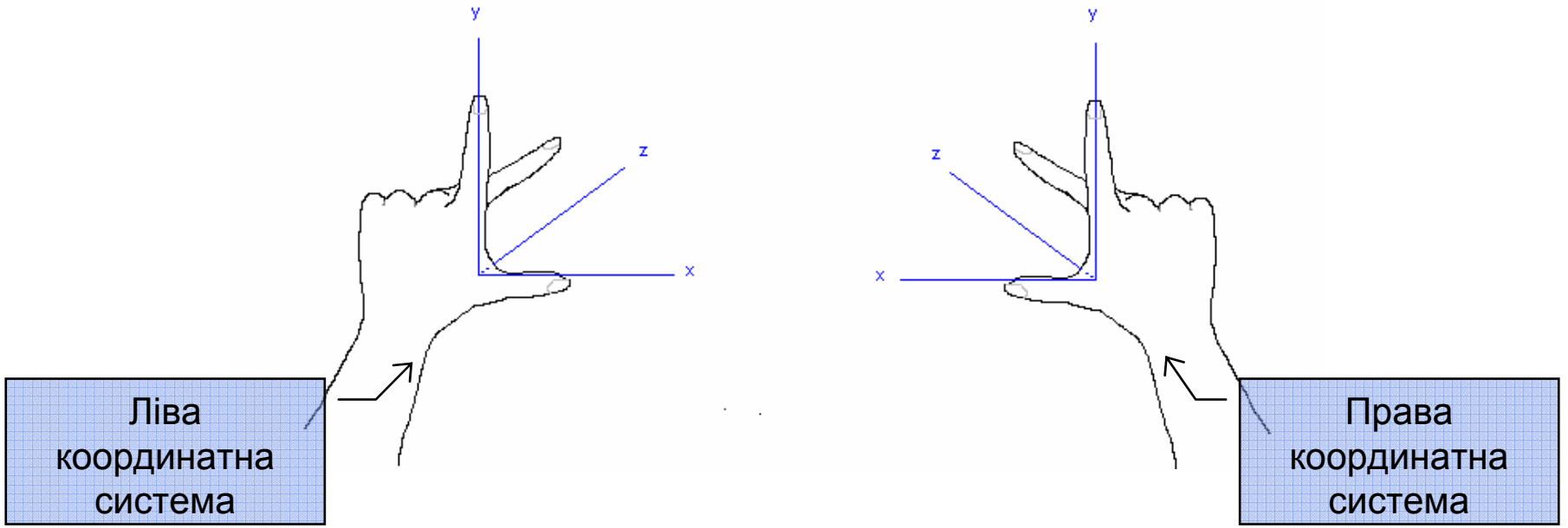


Загально прийнята система координат (0,0,0)

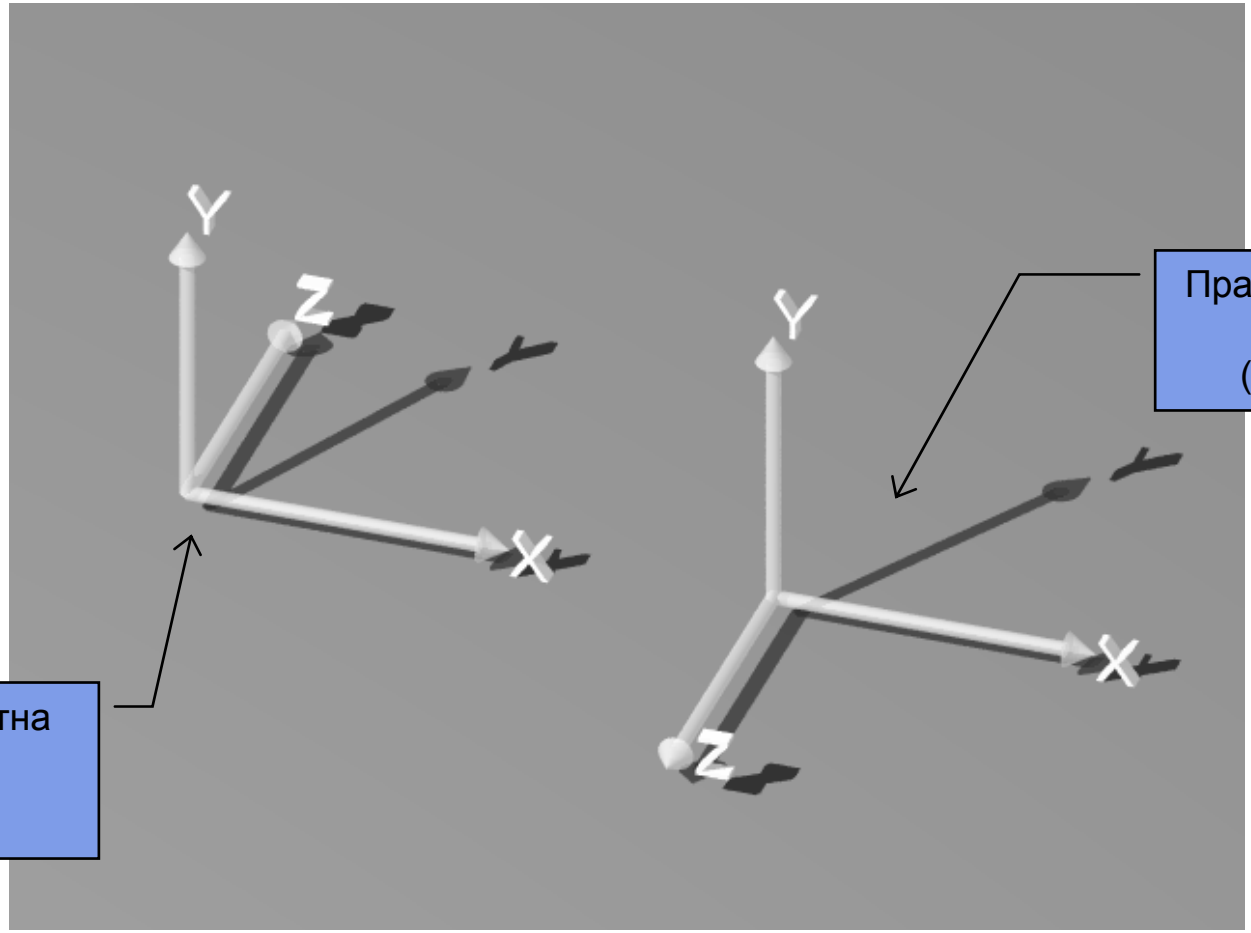
Центр конуса (3,0.2,2)

Координатна система

- В POV Ray використовується правило координатної системи лівої руки
- В стандартній математичній системі використовується правило координатної системи правої руки



Координатна система



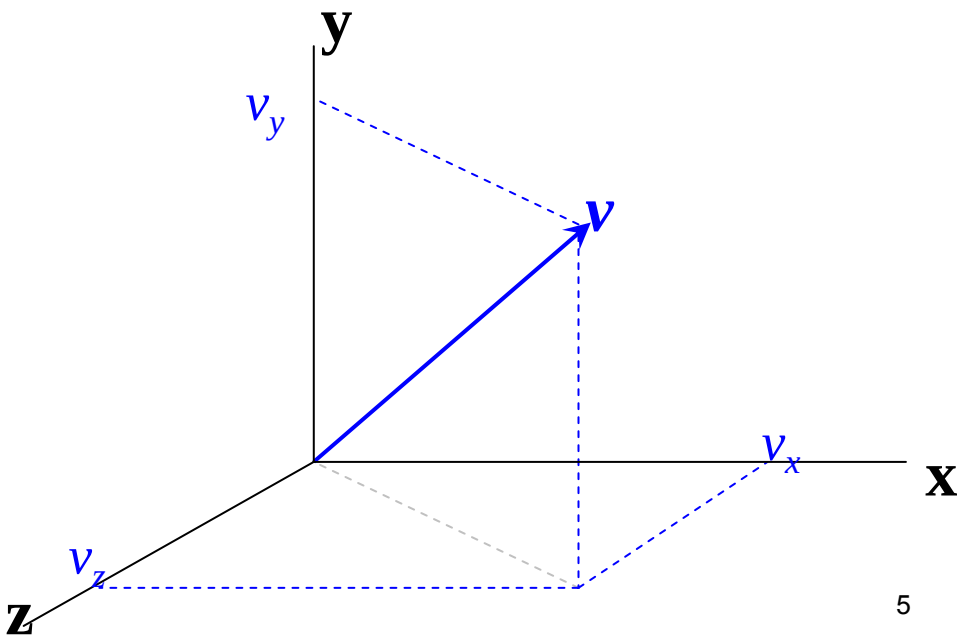
Права координатна система (математика)

Ліва координатна система (POV-Ray)

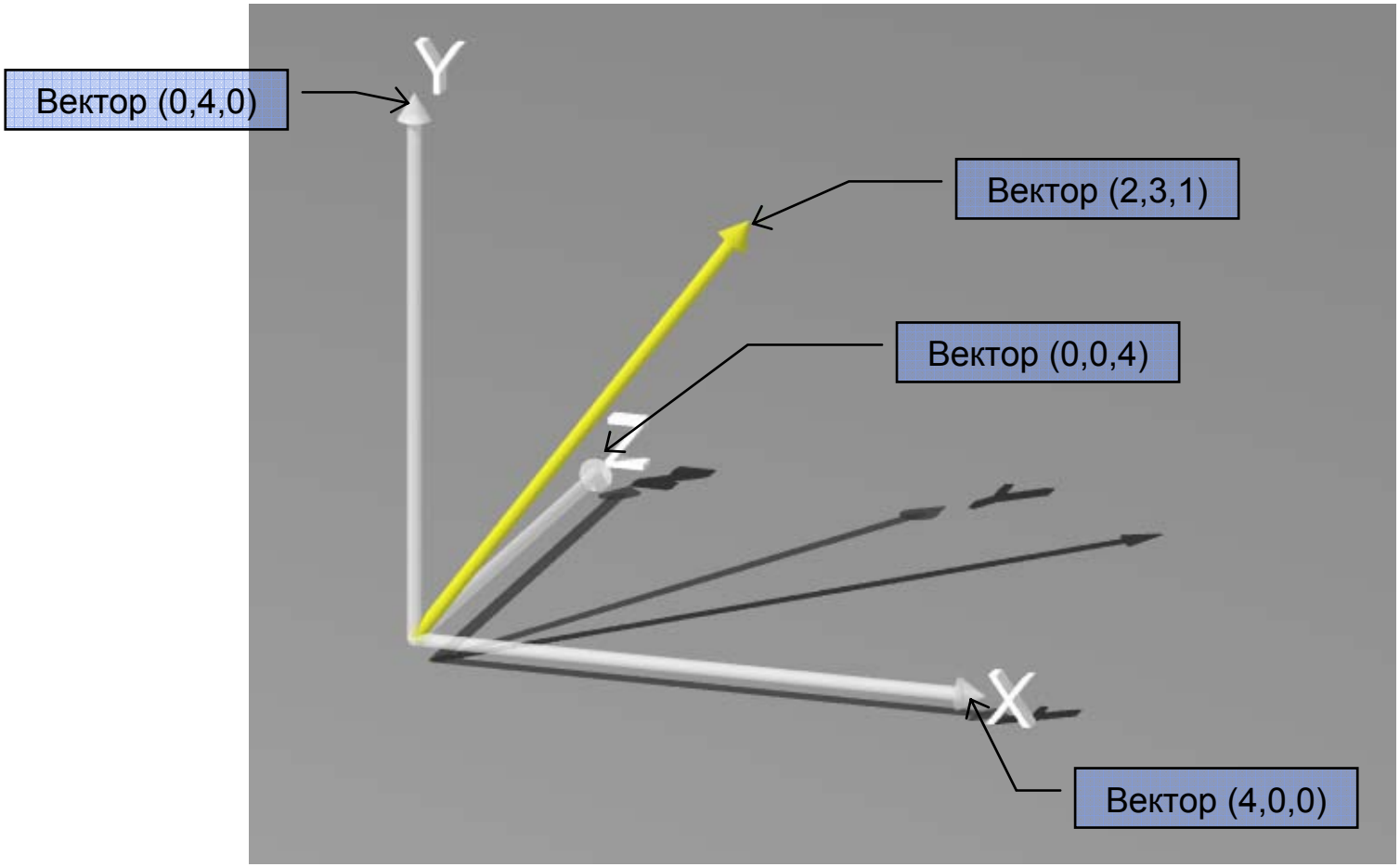
Вектор

- В 3D вектор визначається за 3 скалярними величинами: (v_x, v_y, v_z)

- Довжина вектора: $|v| = \sqrt{v_x^2+v_y^2+v_z^2}$

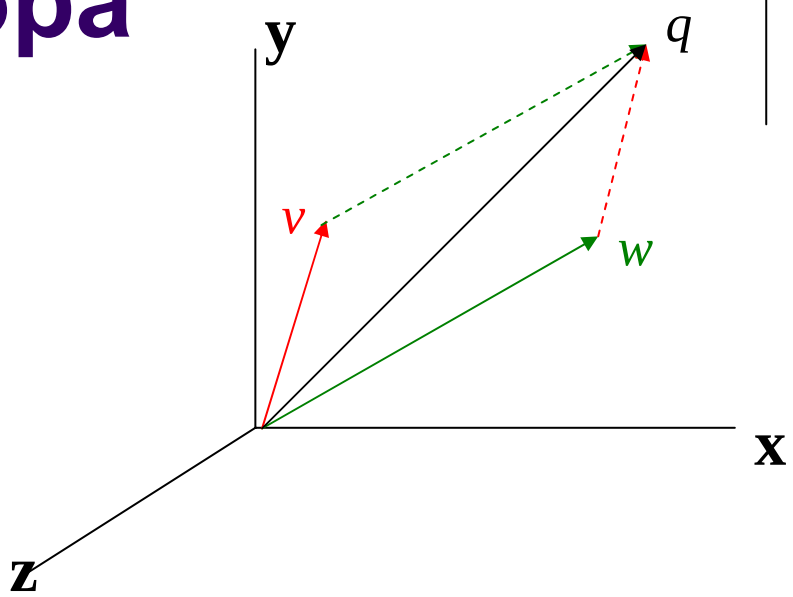


Вектори



Векторна алгебра

- Додавання
  - $q = v + w$

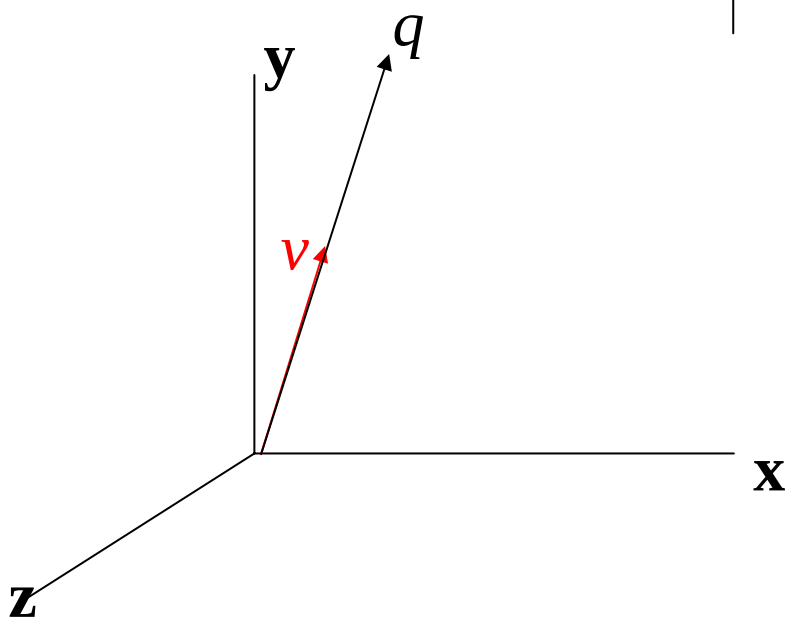


q отримуємо додаванням координат векторів v та w:

$q_x = v_x + w_x$, $q_y = v_y + w_y$, $q_z = v_z + w_z$

Векторна алгебра

- Скаляр
  - Величина скаляра s.
  - $q = sv$



q визначається множенням координат вектора на величину скаляра:

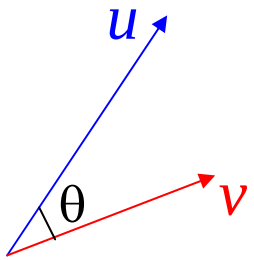
$q_x = sv_x$, $q_y = sv_y$, $q_z = sv_z$

Множення векторів

Беремо 2 вектори: u та v, де:

$$u = (u_x, u_y, u_z)$$

$$v = (v_x, v_y, v_z)$$

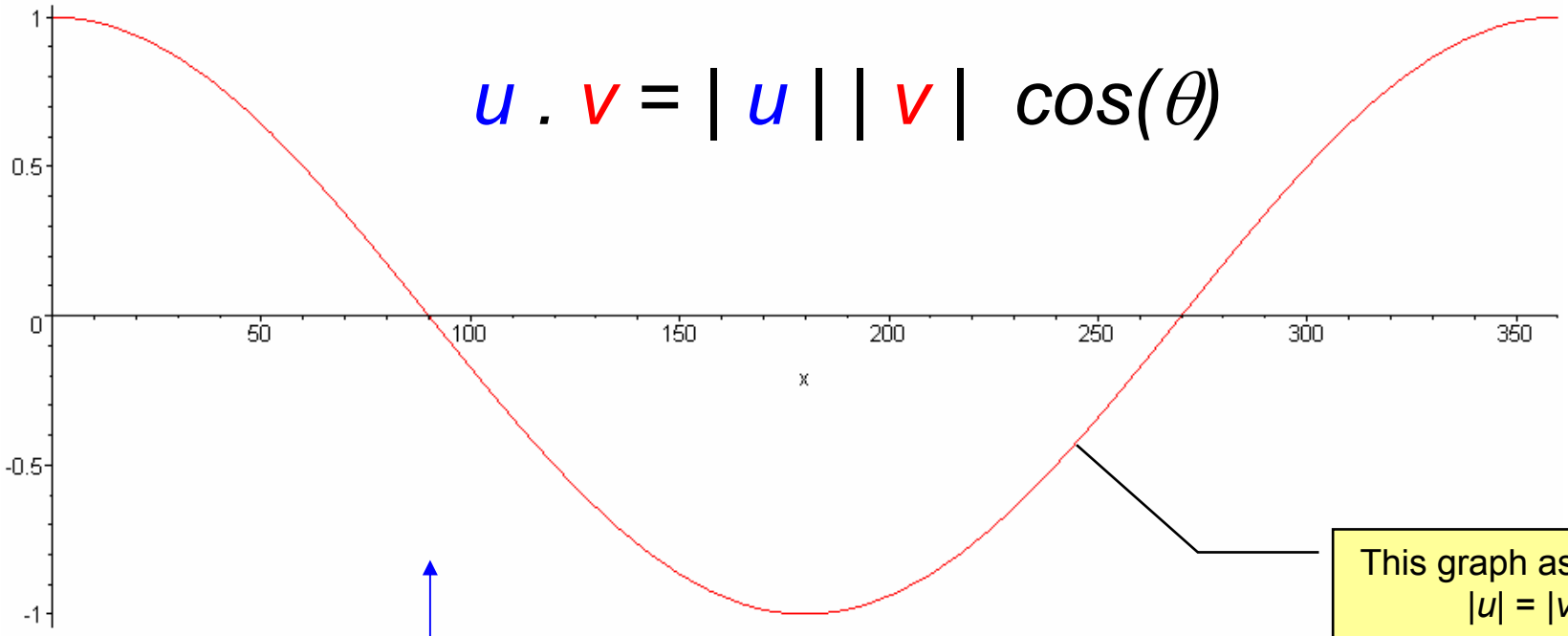


Their dot product (or “inner product”) is a scalar number obtained as the sum of the component-wise products:

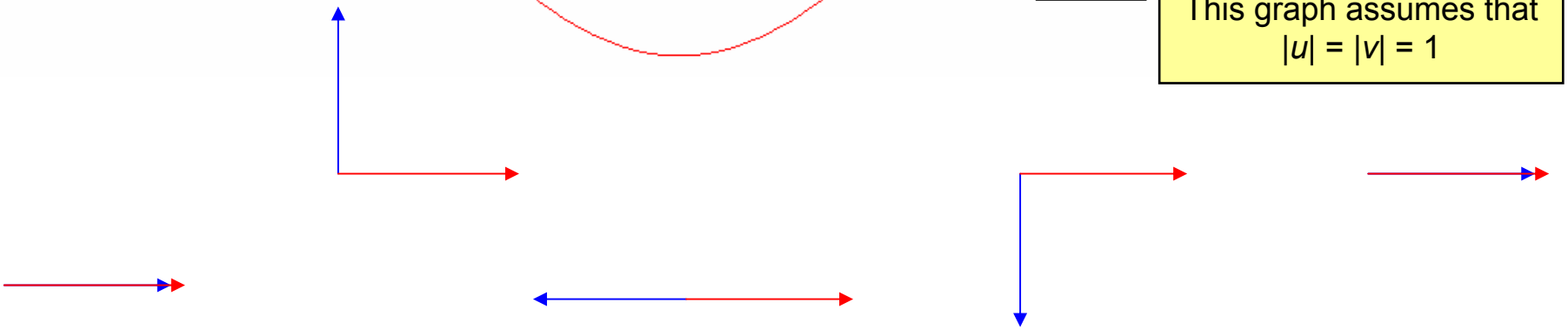
$$u \cdot v = u_x v_x + u_y v_y + u_z v_z$$

Dot Product

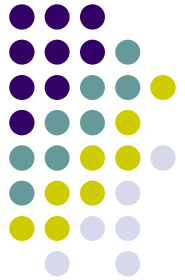
$$u \cdot v = |u||v| \cos(\theta)$$



This graph assumes that $|u| = |v| = 1$

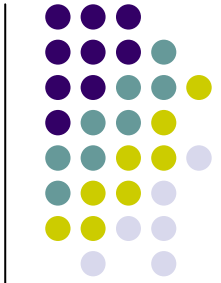


Dot Product



- Properties
  - Symmetric: $u.v = v.u$
  - Bilinear: $v.(u + aw) = v.u + a(v.w)$
  - Non-degenerate: $v.v = 0$ only if $v = 0$.
  - If v and u are orthogonal: $v.u = 0$
  - Positive
    - If angle is less than 90
  - Negative
    - If angle is more than 90

Cross Product



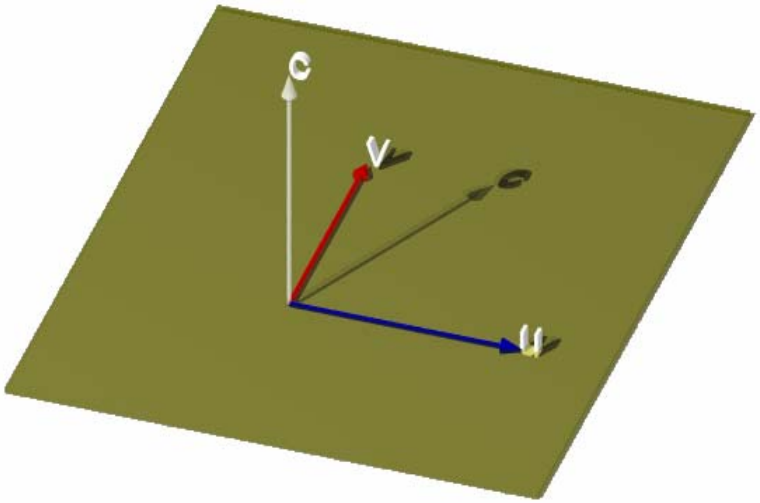
The cross product of two vectors produces a third vector which is perpendicular (orthogonal) to the plane in which the first two lie

Given 2 vectors: u and v, where:

$$u = (u_x, u_y, u_z)$$
$$v = (v_x, v_y, v_z)$$

Their cross product (or "outer product") is a new vector $c=(c_x, c_y, c_z)$ computed as follows:

$$cx = u_y v_z - u_z v_y$$
$$cy = u_z v_x - u_x v_z$$
$$cz = u_x v_y - u_y v_x$$



- Для двох точок, заданих своїми координатами $A(x_1; y_1)$ та $B(x_2; y_2)$, можна розв'язати такі задачі:
  1. Знайти *відстань між двома точками*:
- $$d=\sqrt{(x_2-x_1)^2+(y_2-y_1)^2}$$
  .
  2. Знайти *Координати середини відрізка*:

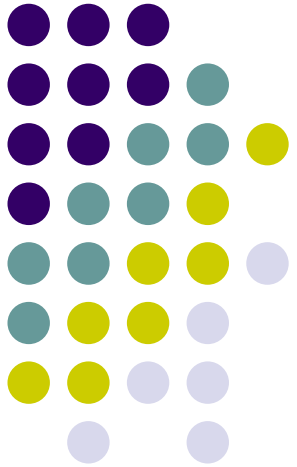
$$C_x=\frac{x_1+x_2}{2};\ C_y=\frac{y_1+y_2}{2}$$

- .
  3. Знайти координати точки D, яка поділяє відрізок AB у *відношенні* $AD:DB=\lambda$:

$$D_x=\frac{x_1+\lambda x_2}{1+\lambda};\ D_y=\frac{y_1+\lambda y_2}{1+\lambda}$$

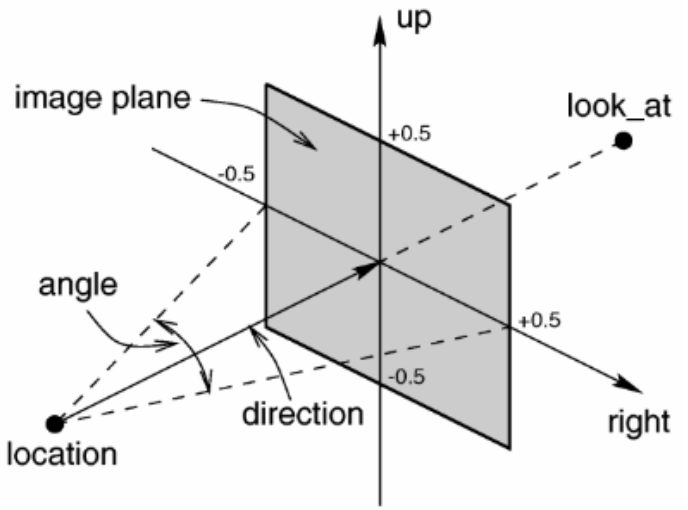
POV-Ray

Більше інформації про мову опису місця дії (scene description language - SDL)



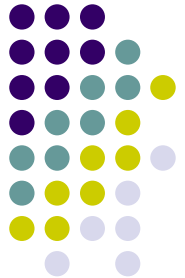
Camera (Камера)

```
camera {
    sky <0,1,0>
    location  <2,  2.2, -3>
    direction <0,  0,  1>
    look_at   <0.7,  1.2,   0>
    up        <0,  3,   0>
    right     <4,  0,   0>
}
```



- *location* та *look_at*: ми вже розглядали
- *sky*: нахил плана
- *up* та *right*: коефіцієнт стиснення зображення (типове значення складає 4/3)
- *direction (*напрям*)*: рідко використовується. Використовується, щоб вказати напрям кута який охоплює місце дії

Прості геометричні об'єкти



- Sphere (Сфера)
- Box (Куб)
- Cylinder (Циліндр)
- Cone (Конус)
- Plane (План)
- Torus (Тора)

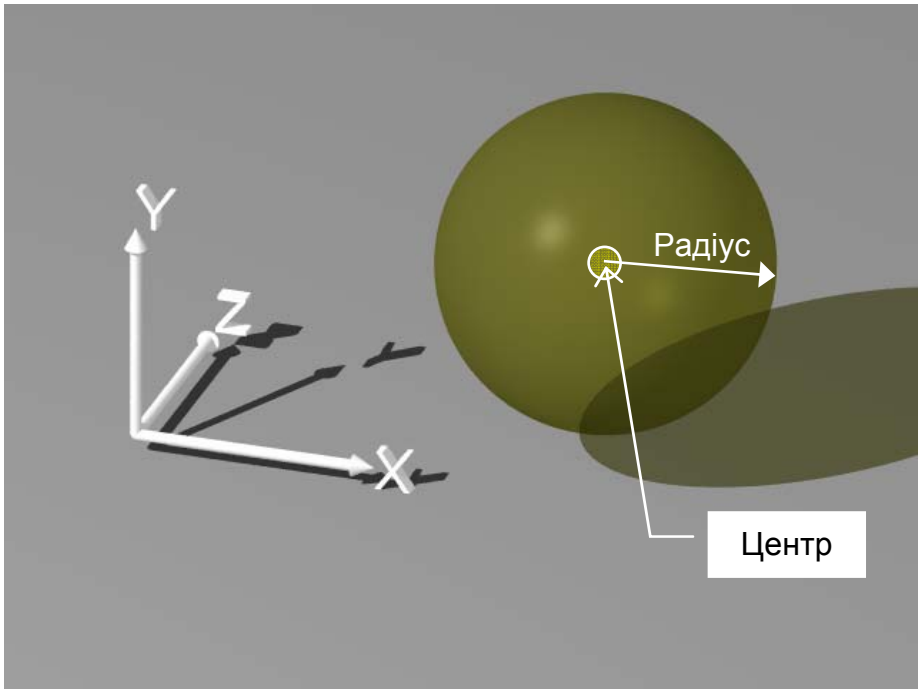
Sphere (Сфера)

```
sphere {
        <Ценр>, Радіус
        //Зовнішні властивості …
    }
```

Центр

```
sphere { <4, 2, 1>, 1.5
         pigment {
                 red 0.9
                 green 0.9
                 blue 0.5
                 filter 0.7}
         finish {
                  phong 0.2
    }
}
```

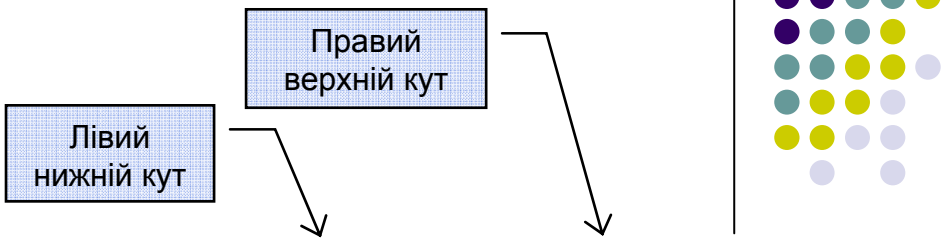
Радіус



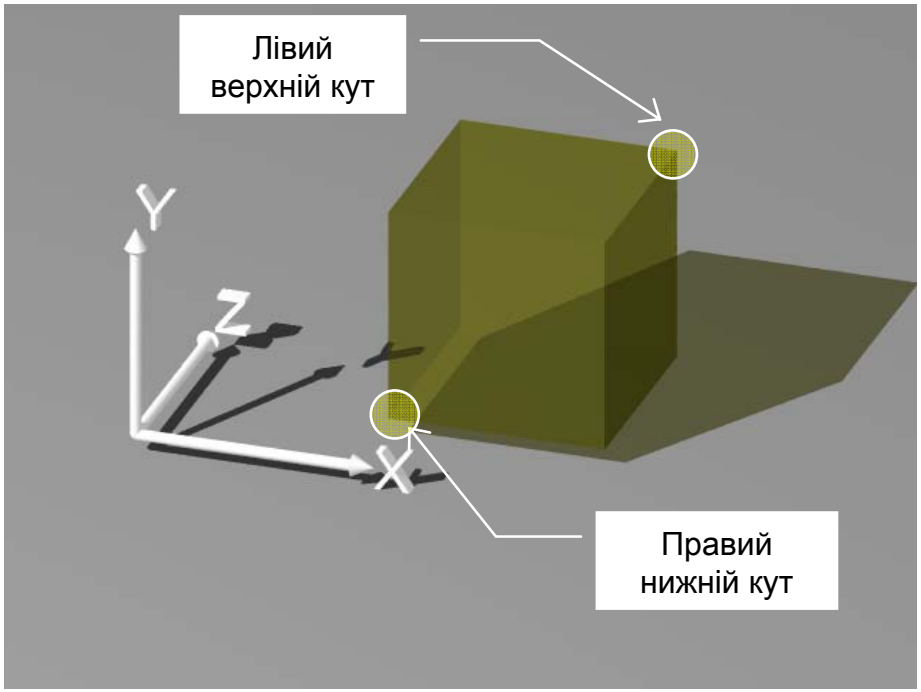
Примітка: Осі маркера в два рази довші у всіх напрямах

Box (Куб)

```
box{
    <лівий нижній кут>, <правий верхній
    кут>

    // Зовнішні властивості …
    }
```




```
box { <2,0.2,1>, <4,2.2,3>
        pigment {
                red 0.9
                green 0.9
                blue 0.5
                filter 0.7}
        finish {
                 phong 0.2
    }
}
```



Примітка: Осі маркера в два рази довші у всіх напрямах

Примітка: Введення координат будь-яких двох протилежних кутів дозволять створити КУБ

Cylinder (Циліндр)

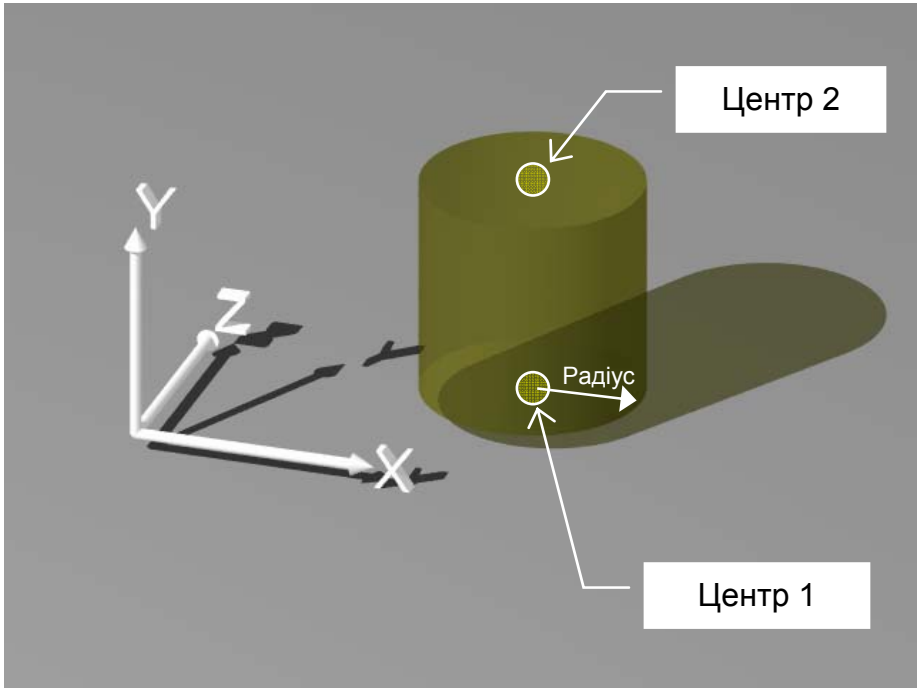
```
cylinder {
    <центр1>,<центр2>,<радіус>
          // Зовнішні властивості …
    }
```

```
cylinder {
          <3,0.2,2>,
          <3,2.2,2>,
          1
        pigment {
                  red 0.9
                  green 0.9
                  blue 0.5
                  filter 0.7}
        finish {
                 phong 0.2
    }
}
```

Центр 1

Центр 2

Радіус



Примітка: Осі маркера в два рази довші у всіх напрямах

Cone (Конус)

```
cone{ <центр1>, <радіус1>,
      < центр2>, < радіус2>
       // Зовнішні властивості…
   }
```

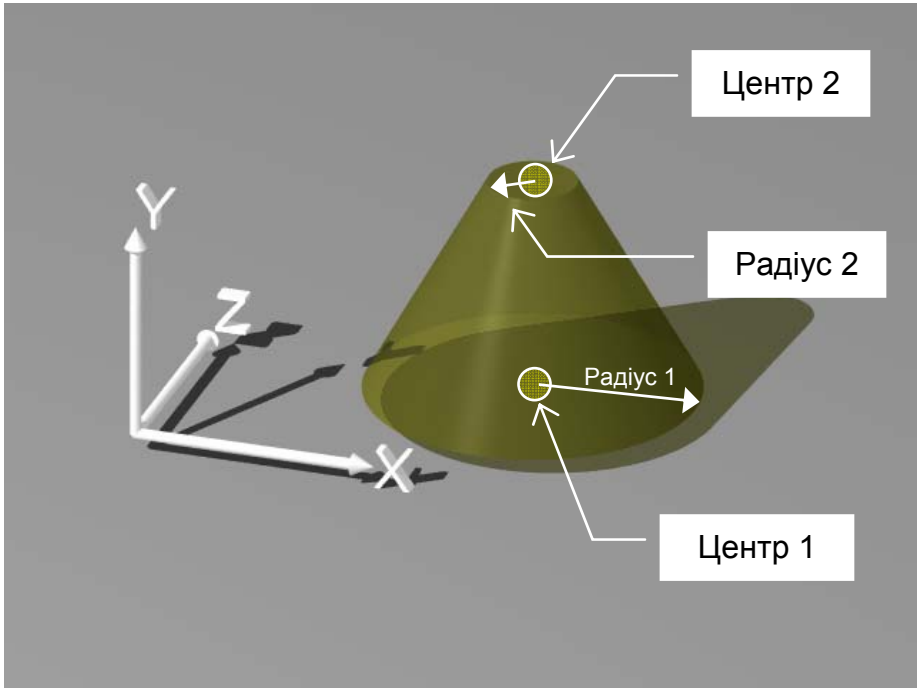
```
cone {
        <3,0.2,2>, 1.5,
        <3,2.2,2>, 0.4
        pigment {
                red 0.9
                green 0.9
                blue 0.5
                filter 0.7
        }
        finish {
                 phong 0.2
        }
}
```

Центр 1

Радіус 1

Радіус 2

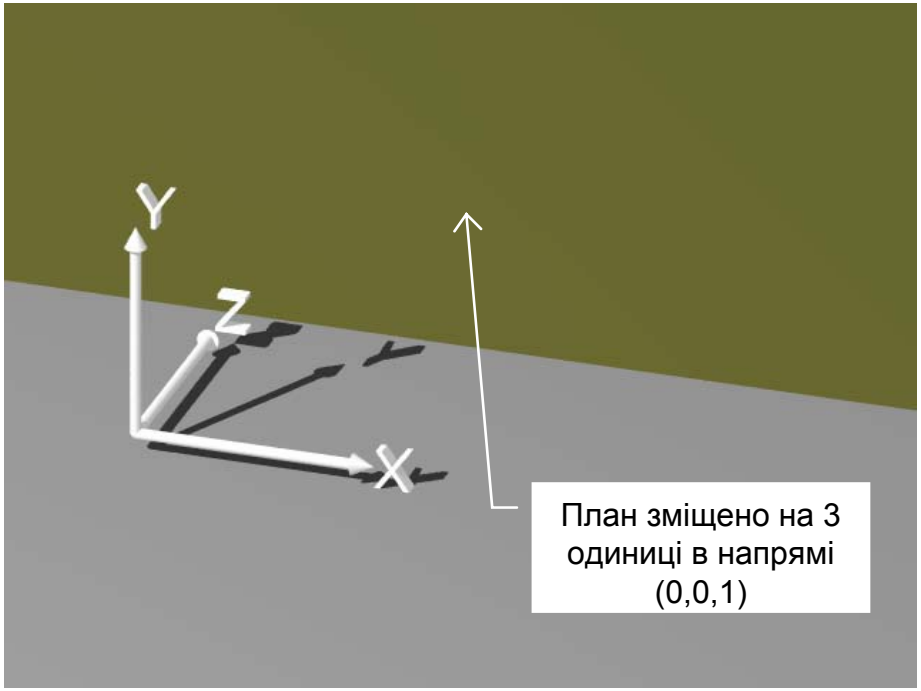
Центр 2



Примітка: Осі маркера в два рази довші у всіх напрямах

Plane (План)

```
plane {<напрям>, <відстань>
        // Зовнішні властивості
      …
   }
```




```
plane {
        <0,0,1>, 3
        pigment {
                red 0.9
                green 0.9
                blue 0.5
                filter 0.7
        }
        finish {
                 phong 0.2
        }
}
```

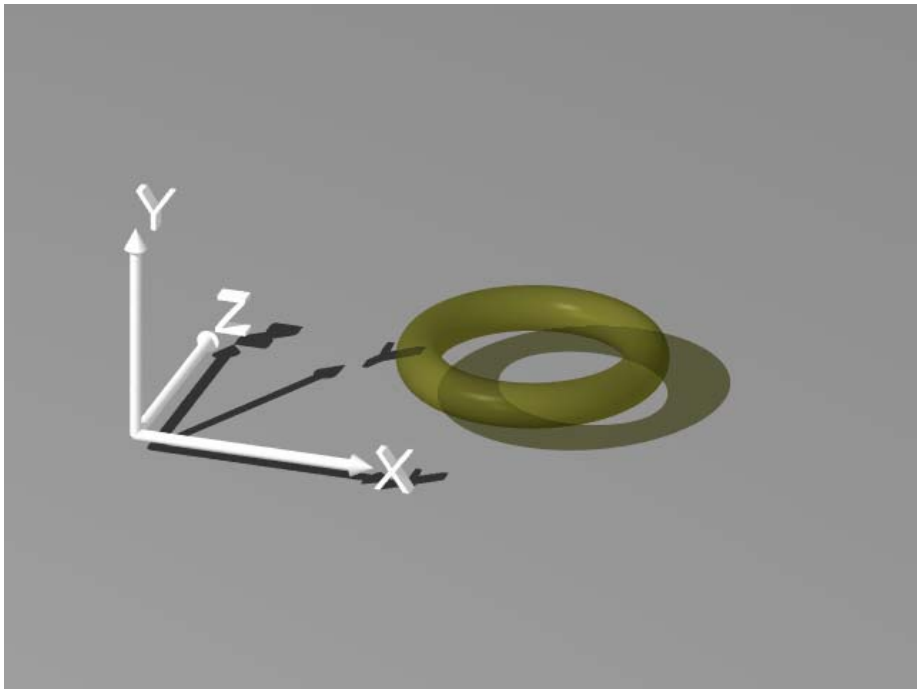
Напрям

Відстань

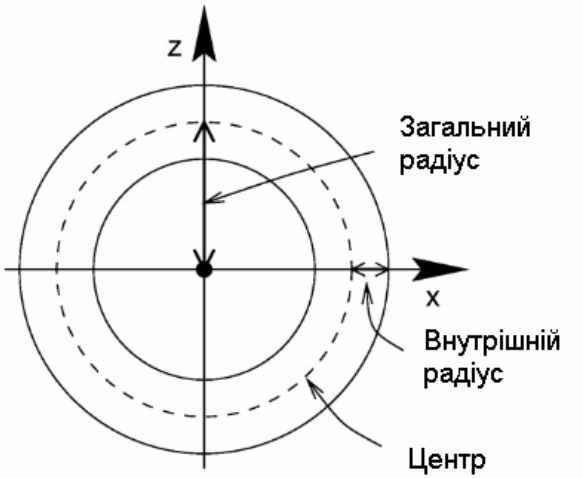
Примітка: Осі маркера в два рази довші у всіх напрямах

Torus (Примітка: Може бути створений тільки в площині x-z)

```
torus{
        Загальний радіус,
        Внутрішній радіус,
    // Зовнішні властивості …
    }
```



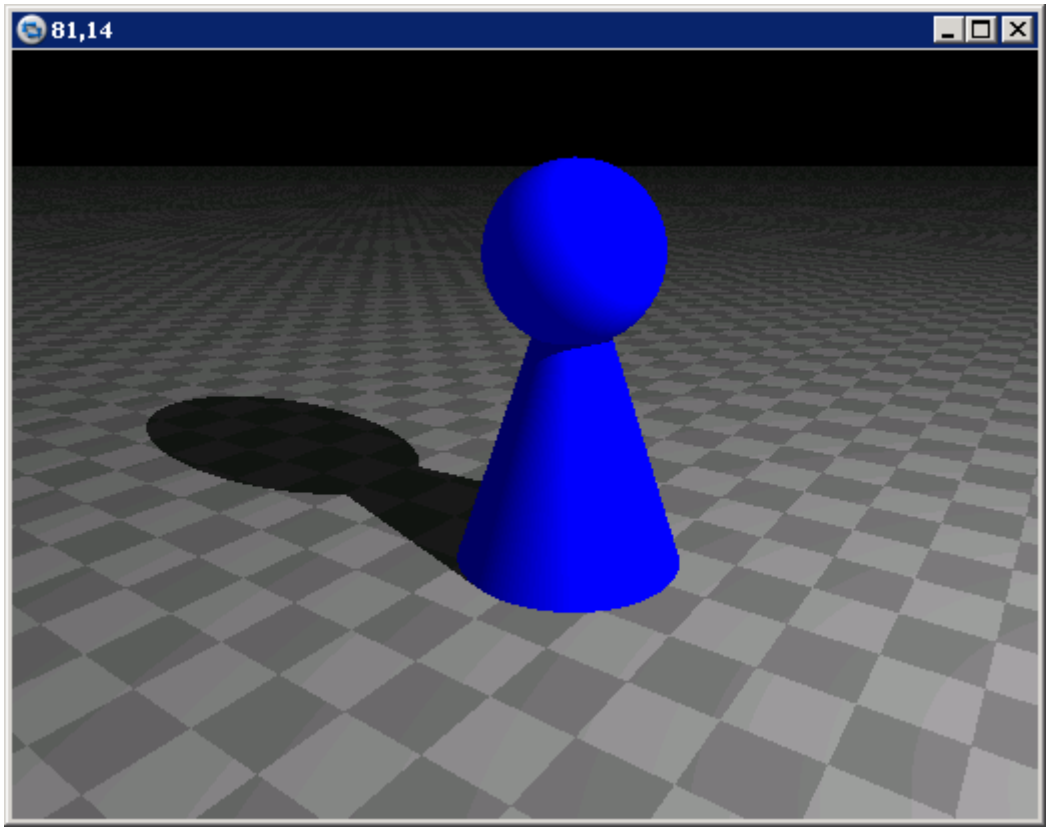

```
torus { 1, 0.2
        pigment {
                  red 0.9
                  green 0.9
                  blue 0.5
                  filter 0.7}
        finish {
                 phong 0.2
         }
        translate <3, 0.5, 2>
}
```



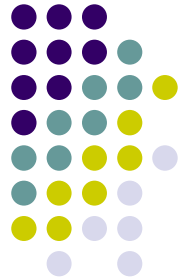
Примітка: Осі маркера в два рази довші у всіх напрямах

Приклад

```
#include "colors.inc“

camera{     location <5, 5, -7>    look_at   <0, 2, 0> }

light_source{ <10, 10, 0> color white }

plane{ <0,1,0>, 0
      pigment{ checker White Gray80 }
}

cone { <0,0,1>, 1.5
       <0,3,1>, 0.5
        pigment { color Blue }
        finish {diffuse 1 ambient .4}
}

sphere { <0,4,1>, 1.1
        pigment { color Blue }
        finish {diffuse 1 ambient .5}
}
```



Вбудовані вектори



SDL має деякі вбудовані вектори, які ми можемо використовувати:

- x, y та z: відповідно <1,0,0>, <0,1,0> та <0,0,1>
- 0: може бути інтерпретований, як <0,0,0>

Наприклад:

```
camera{
   location <5, 5, -7>
   look_at   <0, 2, 0>
}
```

теж саме

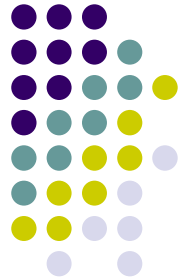
```
camera{
   location <5, 5, -7>
   look_at   2*y
}
```

```
light_source{ <10, 10, 0> color White }
```

Теж саме

```
light_source{ 10*x + 10*y color White }
```

Переміщення, розмірність, розташування



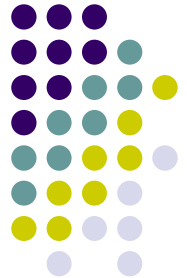
- Translate (переміщення)
- Scale (масштабування)
- Rotation (обертання)

Translate (Переміщення)

- Дозволяє переміщати об'єкт
- Переміщує об'єкт відносно його попереднього місця розташування до руху

```
Translate <a,b,c>
    Переміщує об'єкт на a одиниць по осі x,на b одиниць по осі y,
   та на c одиниць по осі z
Translate 3*x
   Переміщує об'єкт на 3 одиниці вздовж осі x.
```

Scale (Масштаб)

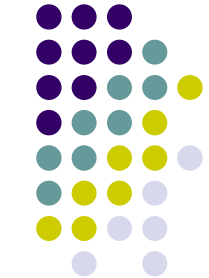


- Дозволяє змінити розмір об'єкта

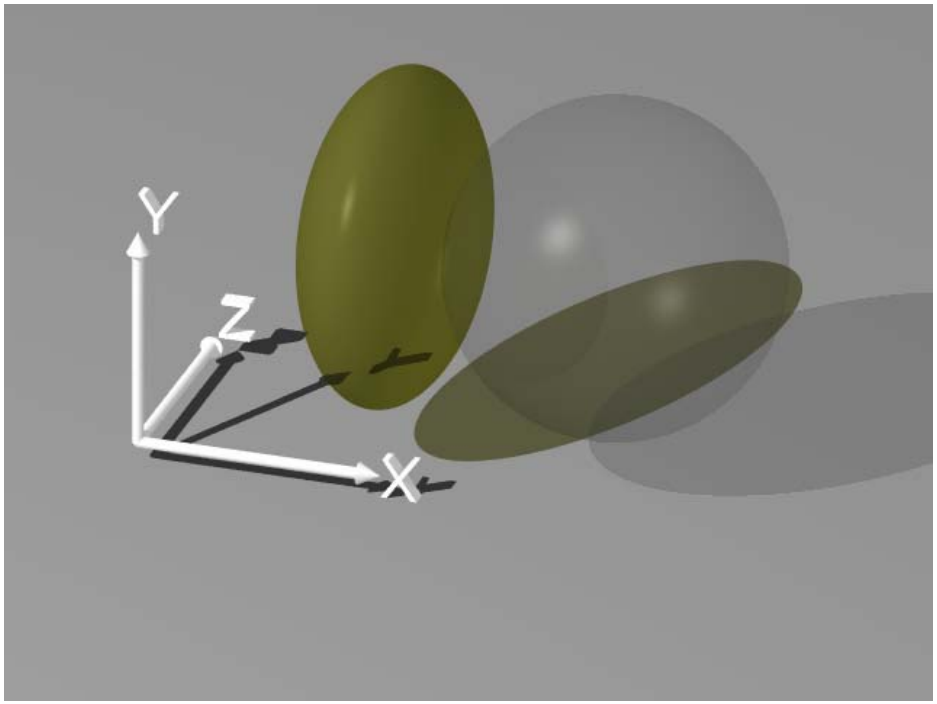
```
Scale <x,y,z>
```

- Три вектора дозволяють змінити масштаб об'єкта в кожному з x, y і z напрямів
- Збільшення відбувається відповідно до Всесвітньо прийнятих правил

Scale (Масштаб)

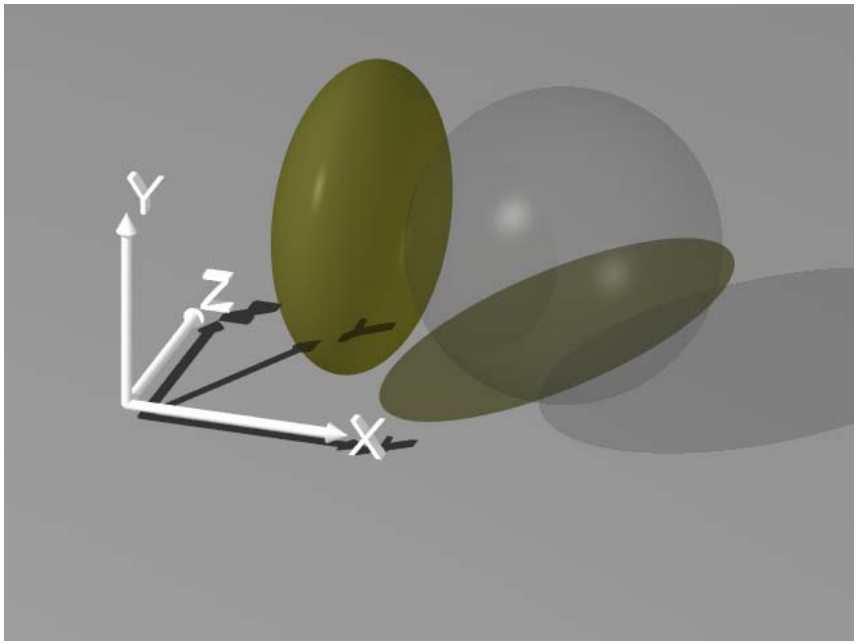


```
sphere { <4, 2, 1>, 1.5
         pigment {
                 red 0.9
                 green 0.9
                 blue 0.5
                 filter 0.7}
         finish {
                  phong 0.2
                }
         scale <0.5,1,1>
}
```

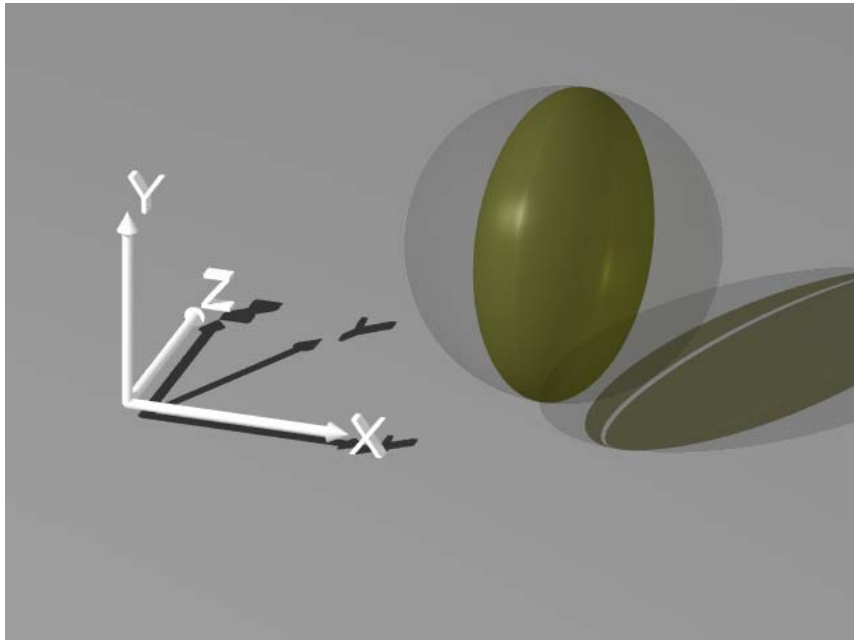


Проблема

Сфера змінила місце розташування після того, як масштабувалась по осі *x* !!!!



Потрібно



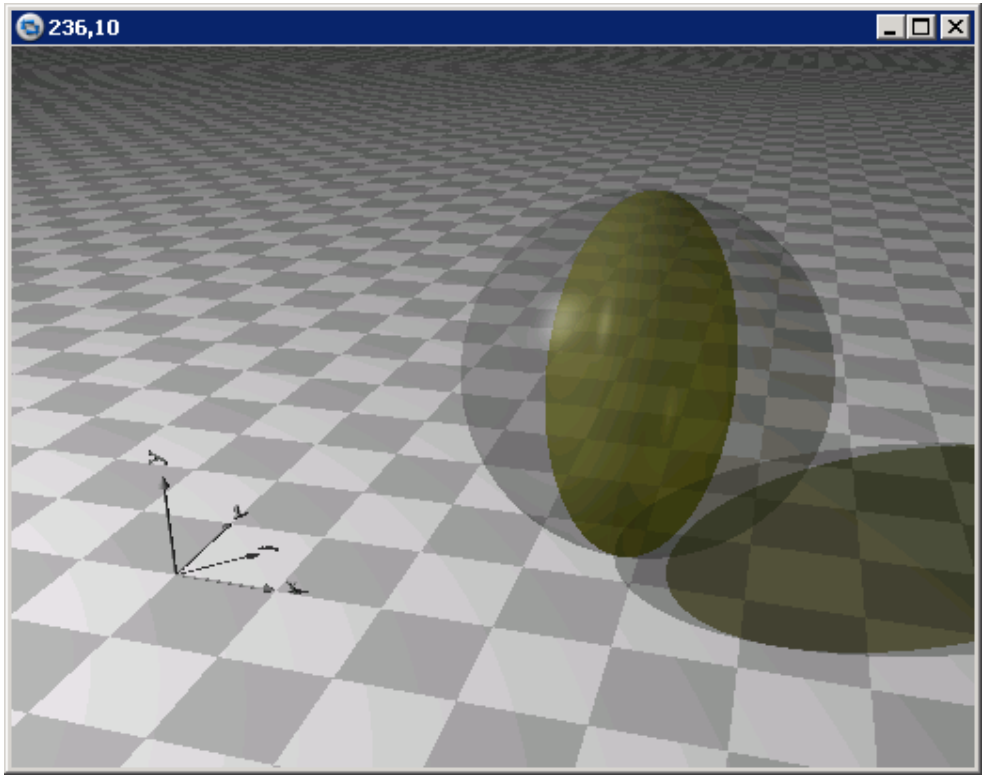
Аналіз:

- Сфера змінила своє місце розташування з координати <4,2,1>
- Пам'ятайте масштабування відбувається відносно оригінального розміру

Аналіз

Правило:

- Створюємо сферу
- Масштабуймо сферу
- Переміщаємо її за координатою <4,2,1>

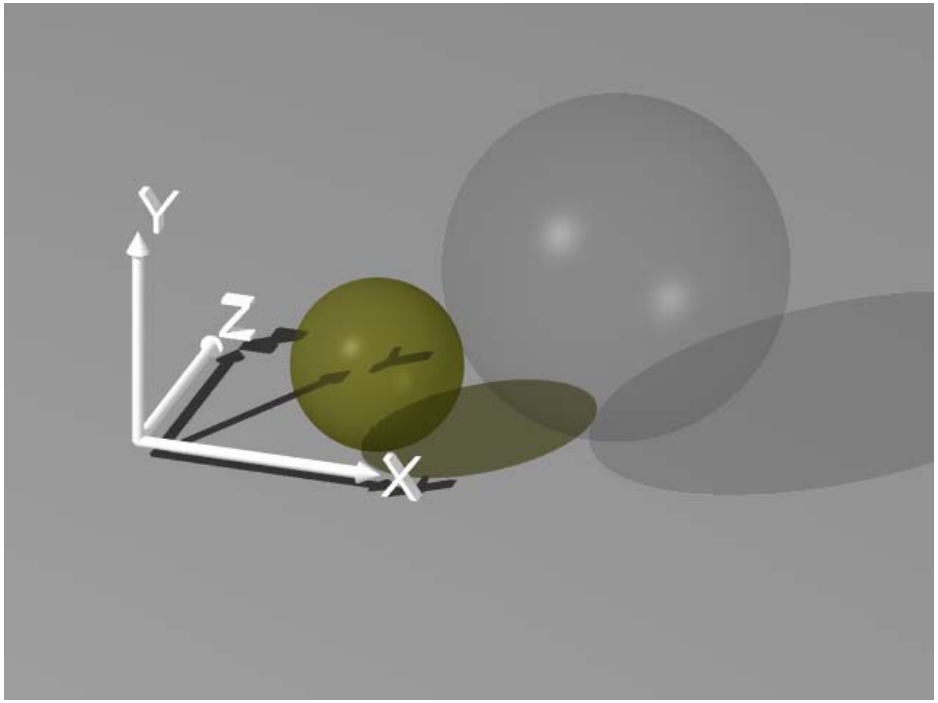



```
sphere { <0, 0, 0>, 1.5
         pigment {
                 red 0.9
                 green 0.9
                 blue 0.5
                 filter 0.7}
         finish {
                  phong 0.2
         }
         scale <0.5,1,1>
         translate <4,2,1>
}
```

Scale (Масштаб)

`Scale n`

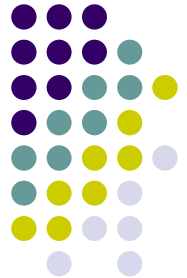
- Розцінюється як <n,n,n> однаково збільшується об'єкт у всіх напрямах




```
sphere { <4, 2, 1>, 1.5
         pigment {
                   red 0.9
                   green 0.9
                   blue 0.5
                   filter 0.7}
         finish {
                  phong 0.2
          }
         scale 0.5
}
```

Примітка:
Масштабування здебільшого застосовується до об'єктів, що складаються з простих об'єктів, а також до складних геометричних об'єктів .

Rotate (Обертання)



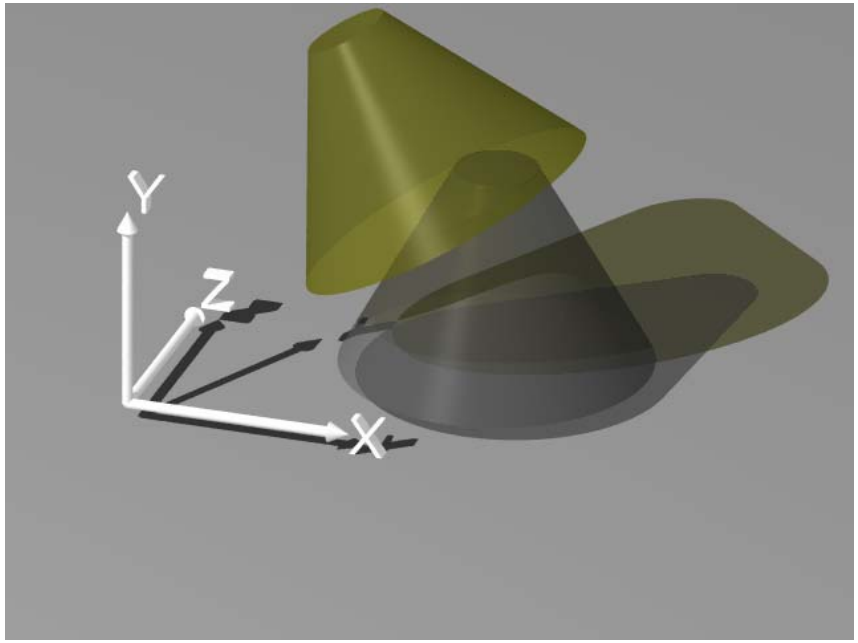
- Змінює орієнтацію об'єкту

```
Rotate <x,y,z>
```

- Обертання об'єкта здійснюється в градусах відносно трьох осей x-, y- та z-
- Обертання відбувається відповідно до Всесвітньо прийнятих правил

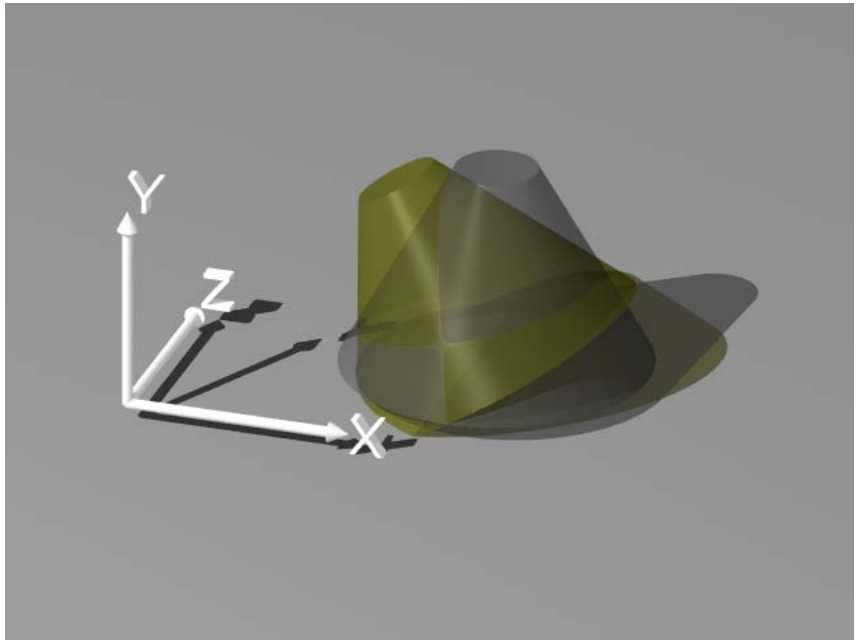
Rotate (Обертання)

```
cone{
        <3,0.2,2>, 1.5,
        <3,2.2,2>, 0.4
         pigment {
                 red 0.9
                 green 0.9
                 blue 0.5
                 filter 0.7}
         finish {
                  phong 0.2
        }
         rotate <0,0,30>
}
```



Rotate (Обертання)

```
cone{
        <0,0,0>, 1.5,
        <0,2,0>, 0.4
         pigment {
                 red 0.9
                 green 0.9
                 blue 0.5
                 filter 0.7}
         finish {
                  phong 0.2
        }
         rotate <0,0,30>
        translate <3, 0.2, 2>
}
```

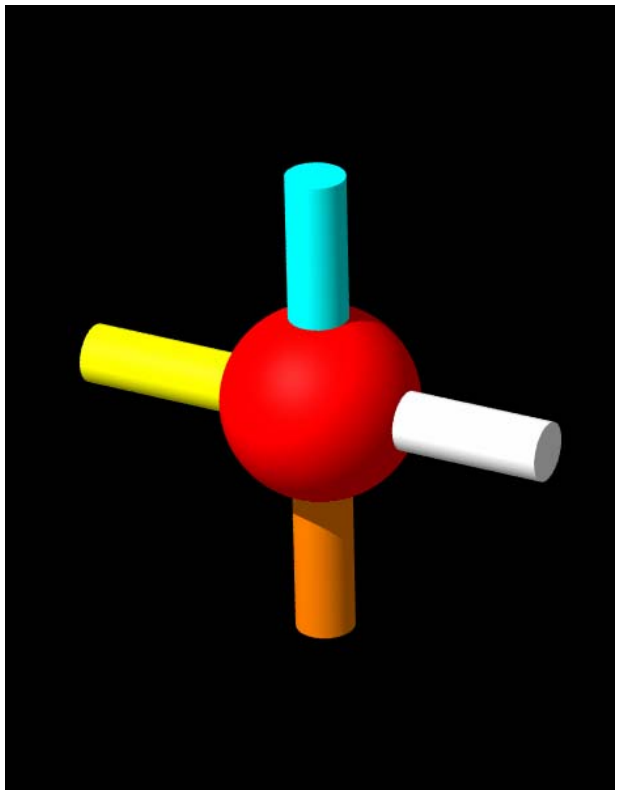


Взагалі

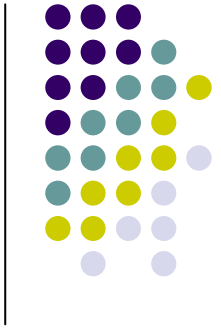
Спочатку необхідно створити об'єкт, а потім застосувати команди:

- Scale
- Rotate
- Translate

Як вправа: Повторіть цю форму:



Text (Текст)



- Об'єкт text створює 3-D текст, як об'єкт літера.

- На даний час POV-Ray підтримує тільки шрифти TrueType font (ttf) та TrueType Collections (ttc)

```
text { ttf "fontname.ttf/ttc" "Текст"
       Товщина, <Початок>
       //Зовнішні властивості…
}
```

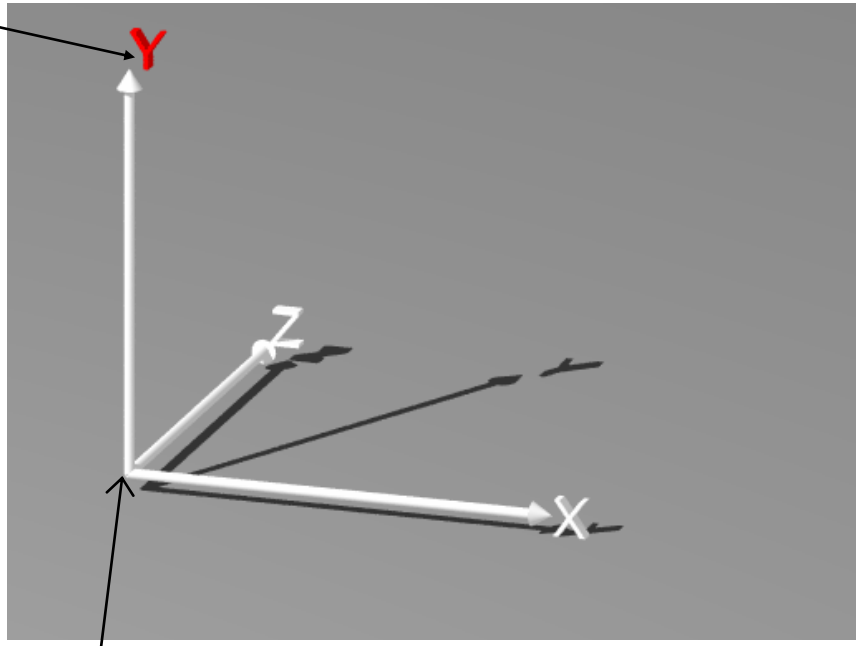
Текст, який виводиться на екран

Початок тексту

Задається товщина літер

Text (Текст)

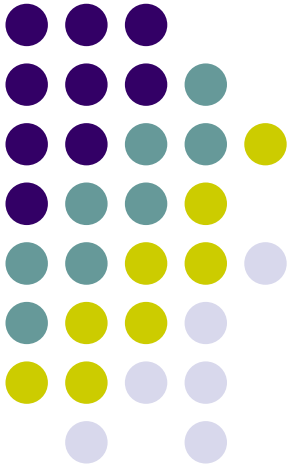
```
text {
     ttf "arial.ttf" "Y" 0.3, 0
     pigment { Red }
     finish {ambient 0.7}
     scale 0.5
     translate 4.2*y
 }
```



Начало <0,0,0>

КС

Конструктивна Стереометрія (КС)



Що таке КС?

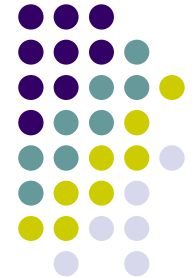
- Дозволяє комбінувати багато простих тіл в форми більш складні, які складаються з простих тіл
- Об'єкти КС можуть бути складені з простих фігур

  Об'єкти КС дозволяють створити більш складні форм

Щоб створити КС об'єкти необхідно:

- Використати #declare, щоб створити форму
- Використати object, щоб показати зразок цієї форми (об'явити)

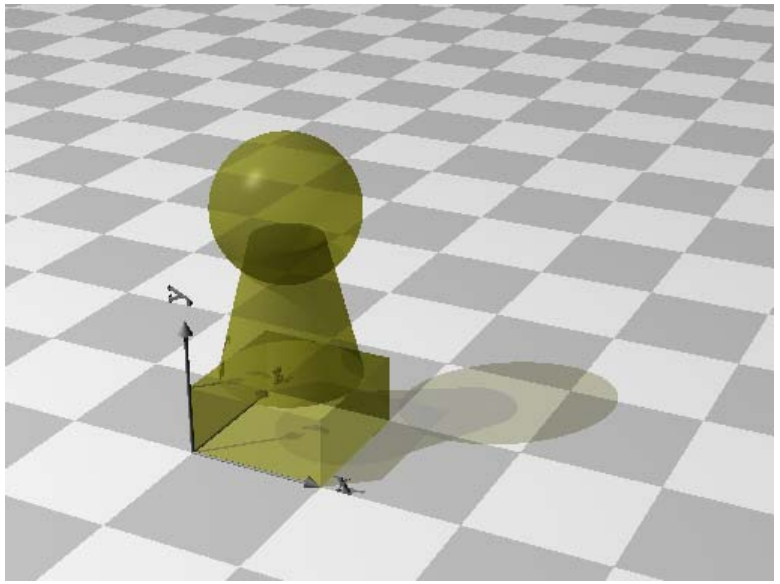
Приклад – Форма об'єднання (union)



```
#declare pawnU = union {
        sphere{<.5, 1.7, .5>, .5}
        cone { <.5,.5, .5>.5, <.5,1.5,.5> 0.25  }
        box { <0,0,0>,<1,.5,1> }
}

object { pawnU
         pigment {
               red 0.9
               green 0.9
               blue 0.5
               filter 0.7}
         finish {
                 phong 0.2   }
 }
```

"pawnU" - це ім'я об'єкта який створений за допомогою форми union (обєднання) і складається з сфери, конуса та куба

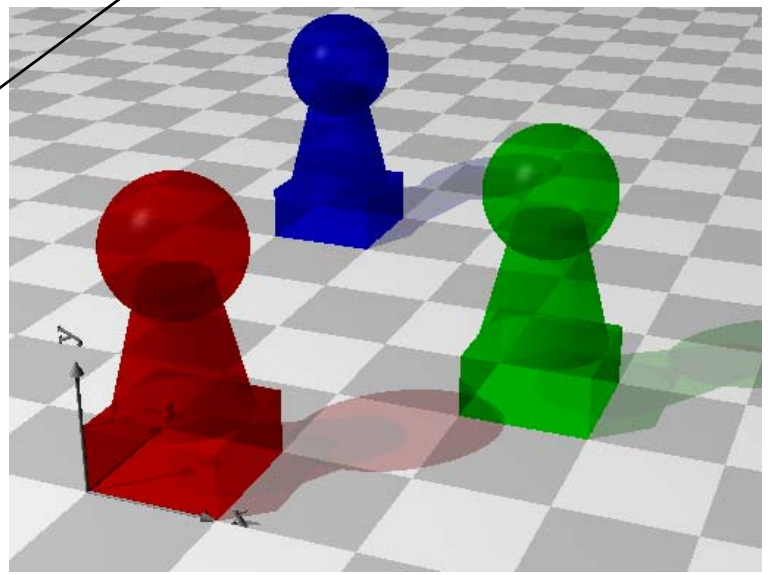
Об'явлення об'єкта "pawnU", що був створений



Збільшення кількості об'єктів

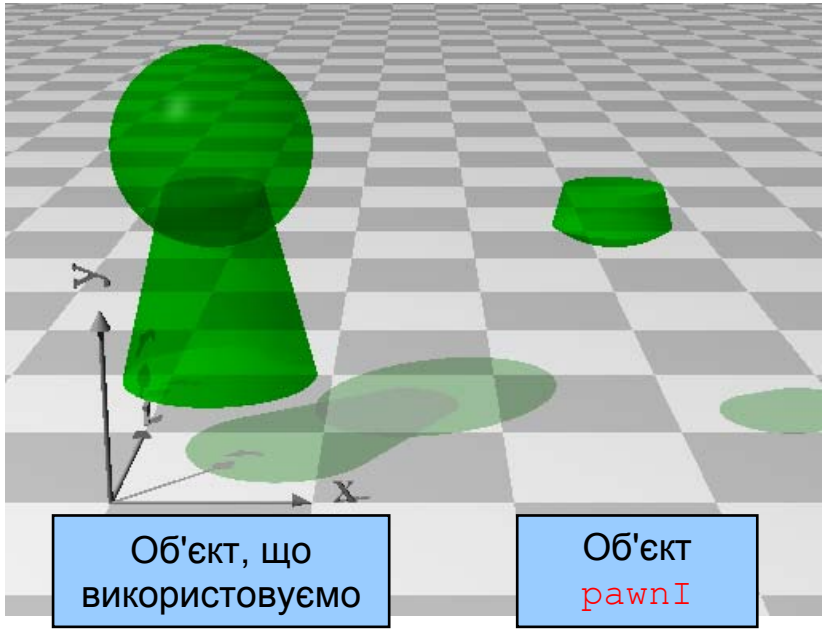
```
object { pawnU
        pigment { color Red filter 0.7}
        finish { phong 0.2   }
 }

object { pawnU
       pigment { color Green filter 0.7}
       finish { phong 0.2   }
       translate <2,0,2>
}

object { pawnU
       pigment { color Blue filter 0.7}
       finish { phong 0.2   }
       translate <-1,0,5>
}
```

Збільшення кількості об'єктів “pawnU”



Приклад - Форма перетин (Intersection)

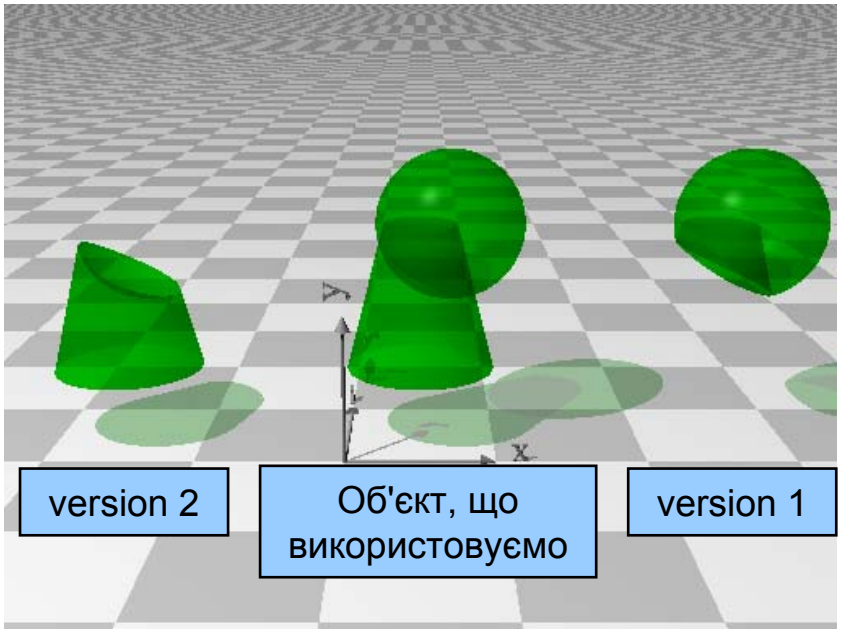
```
#declare pawnI = intersection{
                sphere{<.5, 1.7, .5>, .5}
        cone { <.5,.5, .5>.5, <.5,1.5,.5> 0.25  }
}

object { pawnI
        pigment { color Green filter 0.7}
        finish { phong 0.2   }
        translate <2,0,0>
}
```



Приклад – Форма різниця (difference)

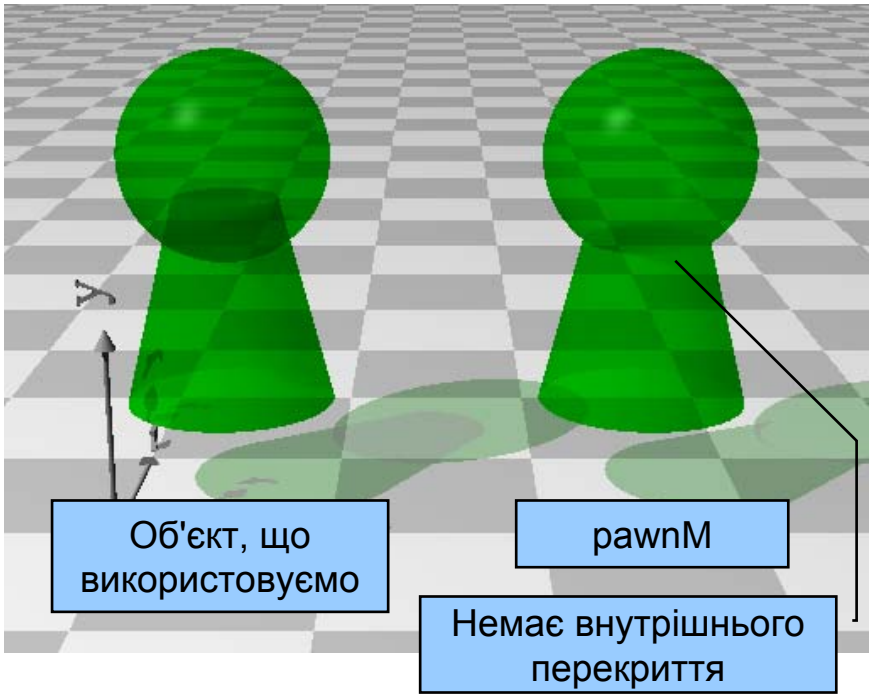
```
// Із сфери віднімаємо конус
#declare pawnD_version1 = difference{
      sphere{<.7, 1.5, .5>,  .5}
      cone { <.5,.5, .5>.5, <.5,1.5,.5> 0.25  }
}

// Із конуса віднімаємо сферу
#declare pawnD_version2 = difference{
      cone { <.5,.5, .5>.5, <.5,1.5,.5> 0.25  }
      sphere{<.7, 1.5, .5>,  .5}
}

object { pawnD_version1
      pigment { color Green filter 0.7}
      finish { phong 0.2   }
      translate <2,0,0>
}

object { pawnD_version2
      pigment { color Green filter 0.7}
      finish { phong 0.2   }
      translate <-2,0,0>
}
```

Примітка:
‘difference’, яка форма вноситься до списку спочатку!



Приклад – Форма злиття (merge)

```
#declare pawnM = merge{
                sphere{<.5, 1.7, .5>, .5}
        cone { <.5,.5, .5>.5, <.5,1.5,.5> 0.25  }
}

object { pawnM
        pigment { color Green filter 0.7}
        finish { phong 0.2   }
        translate <2,0,0>
}
```



Merge працює подібно до *union* за винятком того, що внутрішні перетини зникають.
Merge використовується в КС тільки для того, щоб створити прозорий об'єкт.

Приклад інших КС