TALLINNA ÜLIKOOL
INFORMAATIKA INSTITUUT

Olga Sergejeva
Rühm IF05

ANIMATSIOONID POV – RAY KESKKONNAS
АНИМАЦИИ В СРЕДЕ POV – RAY

Seminaritöö

Juhendaja: lektor Olev Räisa

Tallinn 2008

СОДЕРЖАНИЕ

ВВЕДЕНИЕ

POV-Ray – это аббревиатура от “Persistance of Vision Ray Tracer". Буквальный перевод фразы “Persistance of Vision” - “инерция зрительного восприятия”. POV-Ray – всемирно известный пакет для создания фото-реалистических изображений и отличается он от множества других, прежде всего следующими двумя свойствами: во-первых - он относится к free-ware (бесплатно распространяемым программам) и, во вторых, он представляет собой программу со свободно распространяемым исходным кодом. Так же, большим отличием от многих других программ является способ задания сцены - это некое подобие программы. При этом, если создаешь анимированную сцену, доступна переменная "время", которая изменяется с каждым кадром в пределах, заданных пользователем в конфигурационном файле к этой сцене. Таким образом можно легко создавать анимированные математические модели, основанные на использовании различных функций от времени.

Сложность же создания анимации в POV-Ray среде прежде всего заключается в том, что трудно найти четкое руководство по описанию движения объектов. Раздел Help дает расплывчатое представление. В добавок, для читателя может проблема усугубиться, если он не знает на должном уровне английский язык, так как все учебные приложения написаны именно на английском языке. Частично данная работа написана, чтобы немного расширить круг читателей, интересующихся возможностями POV-Ray пакета. Главная же цель состоит в том, чтобы как можно ближе познакомить читателя с возможностями создания анимационных картин. Для этого познакомимся с основными понятиями, терминами, встречающиеся при создании мультипликационных объектов, рассмотрим переменную часов, многоступенчатую мультипликацию, параметры настройки файла QUICKRES.INI, а также разберем примеры. Также в этой работе затронута проблема по преобразованию фреймов с расширением bmp в файл AVI.

Ознакомившись с данной работой, читатель получит начальный толчок для создания своей первой анимации и, возможно, интерес на создание более сложных мультипликационных объектов.

I ОСНОВНЫЕ ПРИНЦИПЫ АНИМАЦИИ В СРЕДЕ POV-RAY

В более ранних версиях POV-Ray пакета, производящих мультипликацию, создание ряда мультипликаций занимало не мало времени и сил, поскольку все было необходимо делать вручную. Например, мы должны были устанавливать переменную часов для каждой индивидуальной структуры отдельно. Мы могли достигнуть некоторой степени автоматизации при использовании командных файлов или подобных scripting устройств, но тем не менее, все это тоже необходимо устанавливать вручную.

Теперь, наконец, с POV-Ray 3, есть более оптимальное решение. Мы больше не нуждаемся в отдельном сценарии или внешних sequencing программах, потому что несколько простых параметров настройки в файле QUICKRES.INI активизируют внутреннюю последовательность мультипликации, которая заставляет POV-Ray автоматически обращаться с деталями мультипликации.

Фактически, имеется два варианта поддержки мультипликации: те параметры настройки, что вставляем в файл QUICKRES.INI, и те кодовые модификации, что вписываем в файл описания сцены.

1.1 ПЕРЕМЕННАЯ ЧАСОВ

POV-Ray поддерживает автоматически заявленную переменную плавающей запятой, идентифицированная как clock. Даже когда мы не устанавливаем отдельно значение часов и не используем петлю мультипликации, переменная часов все же присваивает по умолчанию значение 0.0. Таким образом возможно установить некоторый код POV для мультипликации, и все еще отдавать это как фотоснимок во время создания нашего объекта.

Самым простым примером в использовании был бы объект, который идет по постоянной прямой, скажем по оси X. В декларации нашего объекта мы имели бы утверждение

```
Translate < clock , 0, 0 >
```

Затем получаем петлю мультипликации.

И это хорошо пока изменяется лишь один элемент или аспект сцены, но что случится, когда мы захотим управлять многократными изменениями в той же самой сцене одновременно?

Для этого нам надо использовать нормализованные значения часов, и затем делать другие переменные в нашей сцене пропорциональными, чтобы хронометрировать. Таким образом, когда мы устанавливаем часы, необходимо сделать так, чтобы они работали от 0.0 до 1.0, и затем использовать это как множитель к некоторым другим значениям. Рассмотрим относительно простой пример:

1.1.1. Катящий шар

```
# include "colors.inc"
camera {
      location < 0, 3, -6 >
      look_at < 0, 0, 0 >
      }

light_source { < 20, 20, -20 > color White }
plane {
      y, 0
      pigment { checker color White color Black }
      }

sphere { < 0, 0, 0 > , 1
pigment {
      gradient x
      color_map {
            [ 0.0 Blue ]
            [ 0.5 Blue ]
            [ 0.5 White ]
            [ 1.0 White ]
              }
      scale .25
      }
      rotate < 0, 0, -clock*360 >
      translate < -pi, 1, 0 >
      translate < 2*pi*clock, 0, 0>
      }
```

В вышеприведенном коде имеются кодовые модификации, с помощью которых задается движение шара. Все это оживить мы сможем настроив параметры в файле QUICKRES.INI (с этим мы познакомимся немного позже).

Предполагая, что рядом структур управляют часы, прогрессивно идущие от 0.0 до 1.0, вышеупомянутый код произведет полосатый шар, который катится слева направо поперек экрана. Здесь мы имеем две цели:

1. Чтобы шар вращался с точной пропорцией к его линейному движению, чтобы было видно, что он катится, а не скользит.
2. Перевести шар из одного пункта в другой.

Для того, чтоб реализовать первую цель, мы начинаем со сферы в прохождении. Поскольку по курсу мультипликации, шар сделает одно полное вращение в 360 градусов, мы используем формулу 360*clock. Так как часы работают от 0 до 1, то вращение пробегов сферы будет 0 до 360 градусов.

После чего перейдем к цели под номером два. Поместим сферу в ее начальную отправную точку. После этого, мы переводим сферу расстоянием относительно часов, таким образом заставляя его переместиться от отправной точки. Мы выбрали формулу 2*pi* r*clock (самая широкая окружность сферы определяет текущее значение часов) так, чтобы шар, казалось, проходит расстояние равное окружности сферы в то же самое время, когда он вращает полные 360 градусов. Таким образом, мы синхронизировали вращение сферы к ее переводу, делая так, чтоб казалось, что сфера гладко катится по поверхности.

Другое серьезное основание для того, чтобы использовать нормализованные значения часов - то, что не будет иметь значение, делаем ли мы десять фреймов, оживляемых GIF, или триста фреймов AVI. Значение часов распределяются к числу фреймов, так, чтобы тот же самый код POV работал независимо от того, какой длины последовательность фреймов. Наш шар, вращаясь, будет все еще путешествовать по тому же самому количеству независимо от того, после какого количества фреймов наша мультипликация заканчивается.

1.2. ПАРАМЕТРЫ НАСТРОЙКИ ФАЙЛА QUICKRES.INI

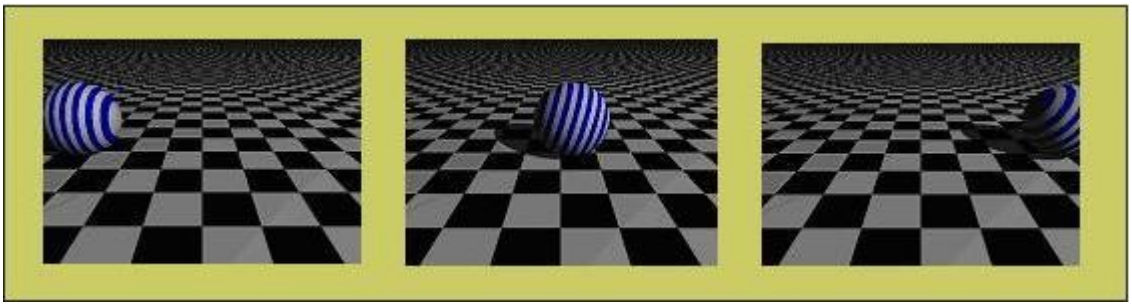
Первое понятие, которое мы будем должны знать, - параметры настройки файла QUICKRES.INI, Initial_Frame и Final_Frame. Они - очень удобные параметры настройки, которые позволят нам отдавать специфическое число фреймов. Для этого

добавляем следующие строки в настройки файла QUICKRES.INI , с помощью которой начнем автоматизировать петлю, которая произведет 20 уникальных фреймов.

```
Initial_Frame = 1
Final_Frame = 20
```

Настройки автоматически прикрепят номер окна к концу любого названия выходного файла, таким образом присваивая каждому окну уникальный номер без необходимости думать об этом. Во-вторых, по умолчанию, это будет периодически повторять переменную часов от 0 до 1, пропорциональных числу фреймов. Это очень удобно, так как, независимо от того делаем ли мы пять фреймов оживляемыми GIF или 300 фреймов последовательность MPEG, мы будем иметь значение часов, которое сделает гладко циклы от точно того же самого начала до точно того же самого конца.

Теперь вернемся к нашему примеру №1. После добавления строк в настройки файла QUICKRES.INI, мы можем запустить нашу программу и наблюдать анимацию (передвижение шара слева направо).



1.3. МНОГОСТУПЕНЧАТЫЕ МУЛЬТИПЛИКАЦИИ

Мы рассмотрели пример, где цель у нас была, чтобы шар катился слева направо. Теперь усложним немного задачу: сделаем так, чтобы для первой половины мультипликации шар все также катился слева направо, затем изменим направление в 135 градусов, чтобы шар начал катиться справа налево, к задней части сцены. Мы должны были бы использовать новые условные директивы предоставления POV-луча, и проверить значения часов, чтобы определить, когда мы достигнем

промежуточного пункта точки, затем начинать отдавать другую подчиненную последовательность часов. Но наша цель, как выше, это, чтобы работать в каждой сцене с переменной в диапазоне от 0 до 1 (нормализованный). Так давайте предположим, что мы держим ту же самую камеру, свет и поверхность, и поставим часы 0т 0 до 2. Для этого добавим к нашему файлу QUICKRES.INI линии

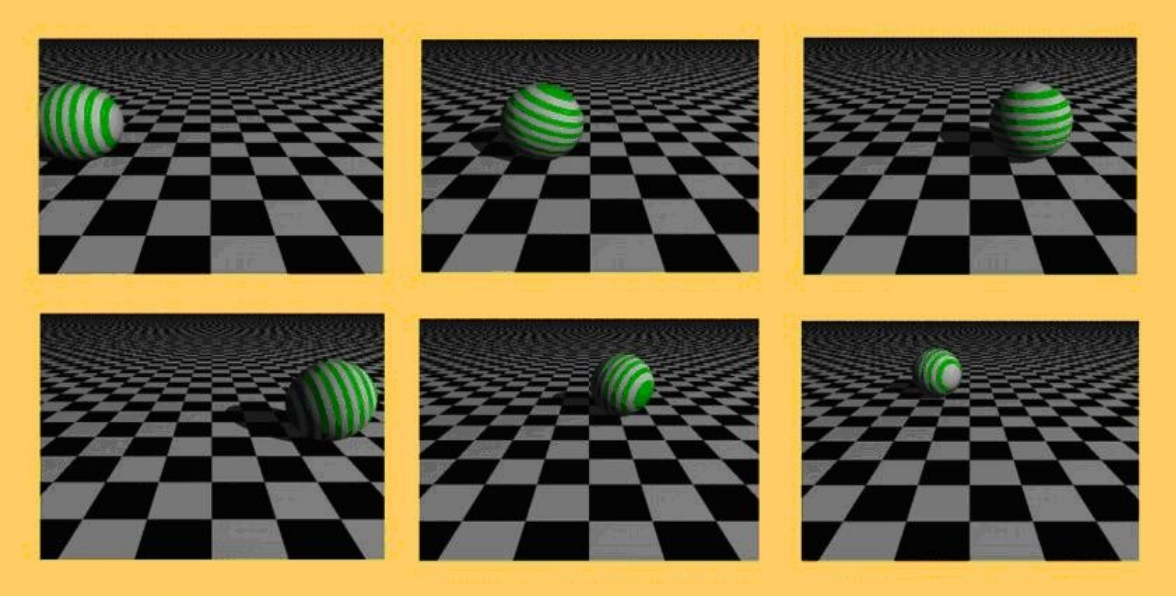
Initial_Clock = 0.0

Final_Clock = 2.0

1.3.1. Движение шара с изменением направления

```
 #include "colors.inc"
camera {
          location <0, 3, -6>
          look_at <0, 0, 0>
          }
light_source { <20, 20, -20> color White }
plane {
          y, 0
          pigment { checker color White color Black }
          }
#if ( clock <= 1 )
     sphere { <0, 0, 0> , 1
     pigment {
               gradient x
                    color_map {
                    [0.0 Green  ]
                    [0.5 Green  ]
                    [0.5 White ]
                    [1.0 White ]
                     }
            scale .25
    }
    rotate <0, 0, -clock*360>
    translate <-pi, 1, 0>
    translate <2*pi*clock, 0, 0>
  }
 #else
  // (if clock is > 1, we're on the second phase)
  // we still want to work with  a value from 0 - 1
  #declare ElseClock = clock - 1;
  sphere { <0, 0, 0> , 1
    pigment {
      gradient x
      color_map {
      [0.0 Green  ]
      [0.5 Green  ]
```

```
      [0.5 White ]
      [1.0 White ]
      }
      scale .25
    }
    rotate <0, 0, ElseClock*360>
    translate <-2*pi*ElseClock, 0, 0>
    rotate <0, 45, 0>
    translate <pi, 1, 0>
  }
#end
```



Все, что мы сделали по-другому, это поменяли промежуток работы часов, чтобы работали бы от 0 до 2. Так, когда часы переходят 1.0, POV предполагает, что вторая фаза поездки началась, и мы объявляем новую переменную Elseclock, которую мы делаем относительно оригинала построенным в часах, таким способом, что, в то время как часы идут 1 - 2, Elseclock идет от 0 до 1. Так, при одном значении clock, могут быть так много дополнительных переменных, сколько мы хотим объявлять. Так даже в довольно сложных сценах, единственная переменная часов может быть сделана общим фактором координирования, который организует все другие движения.

1.4. КЛЮЧЕВОЕ СЛОВО «PHASE»

Есть другое ключевое слово, которое мы должны знать в целях мультипликаций: phase. Ключевое слово phase может использоваться во многих элементах структуры,

особенно в тех, где объекту придается цвет, пигмент или цветовая гамма. Запомним форму образования цветовой гаммы . Например наш флаг:

```
color_map {
  [0.0 White  ]
  [0.33 White  ]
  [0.33 Black ]
  [0.66 Black ]
  [0.66 Blue ]
  [1.00 Blue ]
}
```

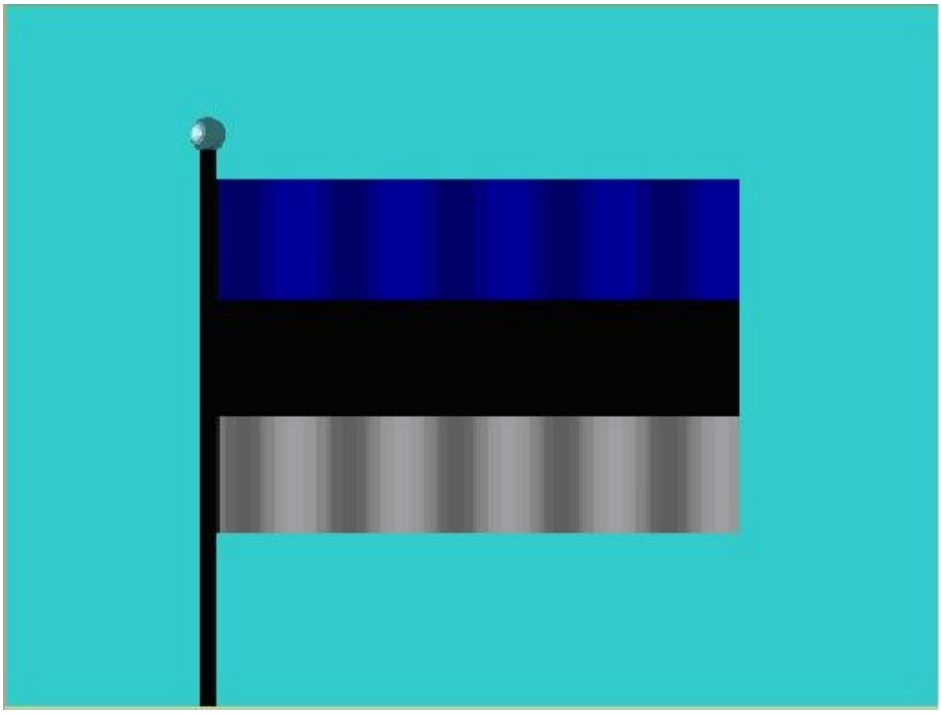
Значение плавающей запятой налево в каждом наборе скобок помогает POV-лучу наносить цвета к различным областям объекта, являющегося текстурованным. Заметьте, что указывается в диапазоне от 0.0 до 1.0.

Фаза заставляет значения цветов изменяться в зависимости от значения плавающей запятой, которая следует за ключевым словом phase. Теперь, если мы используем нормализованное значение часов, мы можем заставить переменную хронометрировать значение плавающей запятой, связанную с фазой, и образец нашего флага гладко перейдет в мультипликацию. Давайте рассмотрим пример:

1.4.1. Развеивание флага

```
#include "colors.inc"
#include "textures.inc"


background { rgb<0.1, 0.8, 0.8> }
camera {
  location <1.5, 1, -30>
  look_at <0, 1, 0>
  angle 10
}
light_source { <-100, 20, -100> color White }
// flag
polygon {
  5, <0, 0>, <0, 1>, <1, 1>, <1, 0>, <0, 0>

pigment {
   gradient y
```

```
    color_map {
      [0.0 White  ]
      [0.33 White  ]
      [0.33 Black ]
      [0.66 Black ]
      [0.66 Blue ]
      [1.00 Blue ]
    }

  }
  normal {
    gradient x
    phase clock
    scale <0.2, 1, 1>
    sine_wave
  }
  scale <3, 2, 1>
  translate <-1.5, 0, 0>
}
// flagpole
cylinder {          <-1.5, -4, 0>, <-1.5, 2.25, 0>, 0.05
  texture {  }
}
// polecap
     sphere {
  <-1.5, 2.25, 0>, 0.1
  texture { Silver_Metal }
}
```



Мы создали флаг с градиентом на нем. Мы вынудили градиент использовать волну типа волны синуса так, чтобы это было похоже, что флаг легко развевается. Здесь мы использовали ключевое слово phase. Для этого надо взять переменную часов как значение плавающей запятой, которая заставит выпуклости и углубления волны флага изменяться по оси X. Фактически, когда мы оживляем фрейм, созданные этим кодом, это будет похоже, что флаг фактически слегка колеблется на ветре.

Это - только один простой пример того, как подчиненное изменение фазы часов может создать интересные эффекты мультипликации. Во время фазы будет видны все виды образцов структуры, и это удивительный диапазон эффектов мультипликации, которые мы можем создать просто одной фазой, фактически не перемещая объект.

II ТИПИЧНЫЕ ЗАДАНИЯ

2.1. ПЕРЕМЕЩЕНИЕ ОБЪЕКТА С МОДИФИКАЦИЯМИ

Ранее мы рассмотрели пример, как шар катился слева направо, после чего менял направление в 135 градусов и начинал двигаться в противоположное направление, к задней части сцены. Попробуем немного модифицировать задание.

Добавим в правый край нашей сцены стенку, для того, чтобы шар, катящийся слева направо, дойдя до этой самой стены, оттолкнулся и начал движение в другое направление. Для сложности, сделаем так, чтобы после отталкивания от стену, шар поменял ни только направление своего движения, но также изменилась сама расцветка шара.

Решение поставленного задания можно увидеть в нижеприведенном POV коде. Также нельзя забывать про настройки файла QUICKRES.INI. В данном случае их оставим прежними:

Initial_Clock = 0.0
Final_Clock = 2.0

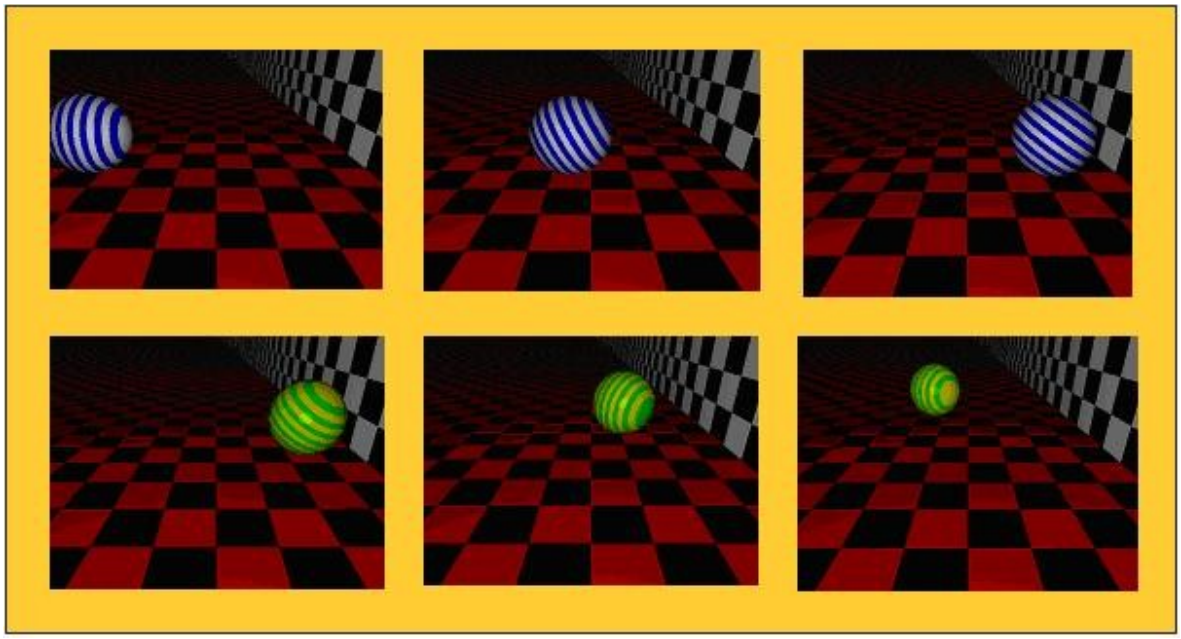
Initial_Frame = 1

Final_Frame = 20

Кодовые модификации, что вписываем в файл описания сцены:

```
# include "colors.inc"
camera {
        location < 0, 3, -6 >
        look_at < 0, 0, 0 >
       }
light_source { < 0, 3, -6 > color White }
plane {
        y, 0
        pigment { checker color Red color Black }
       }
plane {
```

```
      <1, 0, 0>, 4
      pigment { checker color White color Black }
     }

 #if ( clock <= 1 )
  sphere { <0, 0, 0> , 1
    pigment {
      gradient x
      color_map {
        [0.0 Blue  ]
        [0.5 Blue  ]
        [0.5 White ]
        [1.0 White ]
      }
      scale .25
    }
    rotate <0, 0, -clock*360>
    translate <-pi, 1, 0>
    translate <2*pi*clock, 0, 0>
  }
#else
  // (if clock is > 1, we're on the second phase)
  // we still want to work with a value from 0 - 1
  #declare ElseClock = clock - 1;
  sphere { <0, 0, 0> , 1
    pigment {
      gradient x
      color_map {
        [0.0 Green  ]
        [0.5 Green  ]
        [0.5 Yellow ]
        [1.0 Yellow ]
      }
      scale .25
    }
    rotate <0, 0, ElseClock*360>
    translate <-2*pi*ElseClock, 0, 0>
    rotate <0, 45, 0>
    translate <pi, 1, 0>
  }
#end
```



В данном примере мы сделали некоторые модификации к примеру № 2, добавив стенку. Таким образом, когда шар проходит первую фазу, он наталкивается на стену, оттолкнувшись от которой автоматически меняется окраска шара. Также как и в ране приведенном примере, здесь используем такую переменную, как Elseclock. Таким образом, используя одну- две переменных, можно создать достаточно сложную и красивую анимацию.

2.2. ПЛАНЕТАРНАЯ СИСТЕМА

Создадим более интересную мультипликацию. В центр нашей сцены поставим Солнце, которое будет вращаться вокруг своей оси. А вокруг солнца будут двигаться несколько планет, которые в то же самое время еще делают вращение вокруг своей оси. Для красочности мультипликации, создадим звездный фон.

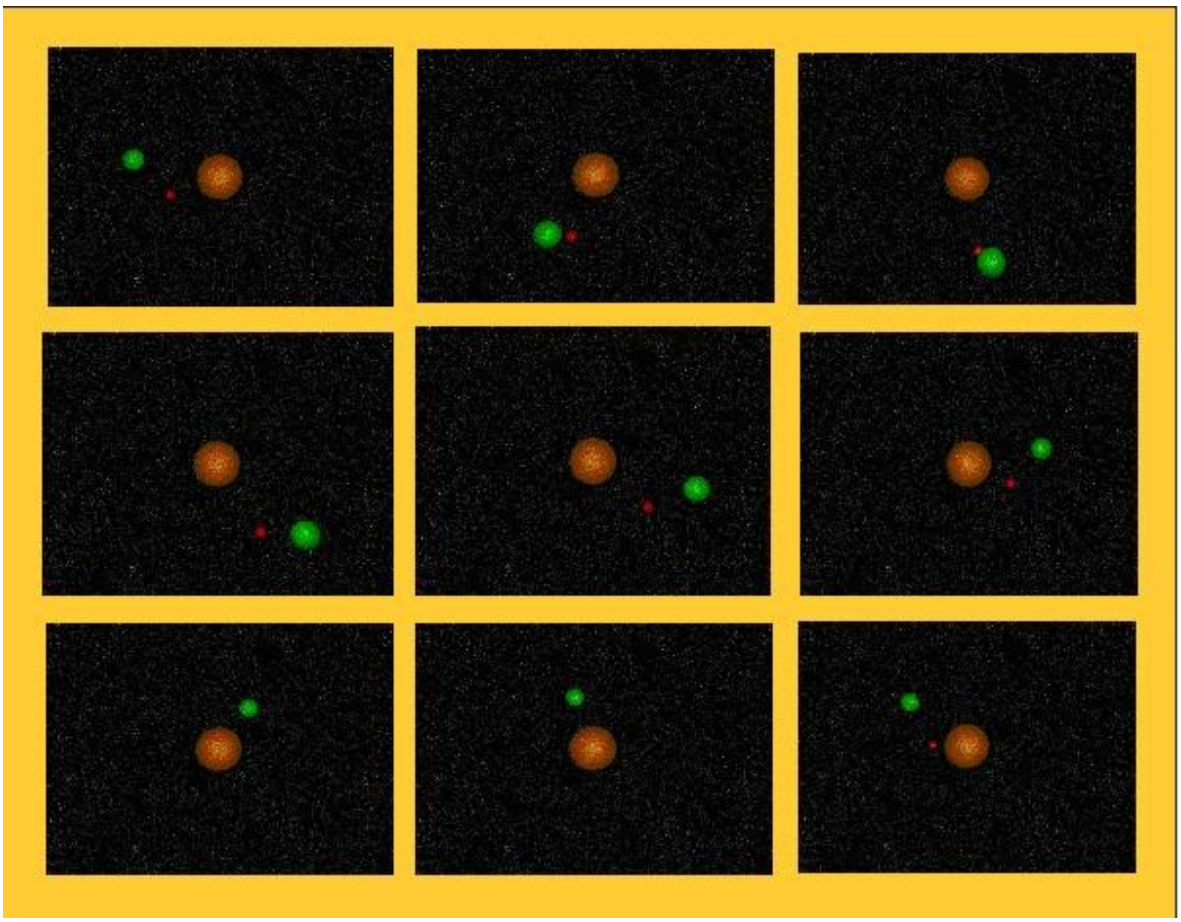
QUICKRES.INI

```
        Initial_Frame = 1
        Final_Frame = 100

        Initial_Clock = 0.0
         Final_Clock = 2.0
```

```
# include "colors.inc"
# include "stars.inc"

  // звездное небо
```

```
sphere { <0,0,0>, 1
        texture { // pigment{ Black}
                  Starfield1 scale 1//  1, 2, ... , 6
                } // end of texture
        scale 20

      }


camera {
       location < 0, -8, -8  >
       look_at < 0, 0, 0 >
      }

light_source { < 0, -6, -6 > color White }

  // солнце
   sphere { <0, 0, 0> , 1

    texture {
          pigment { color Orange }

          normal{bumps 0.9 scale 0.03}
          finish{ phong 1

          }
        }
        rotate <0, clock*360, 0>       // вращение солнца вокруг своей оси
   }

// зеленая планета
   sphere { <0, 4, 0> , 1
     texture {
          pigment { color Green }

          normal{bumps 0.9 scale 0.03}
          finish{ phong 1
            }
        }
     scale 0.5
     translate <-4*cos(360*clock*pi/180.0), -2-4*cos(360*(1-clock*pi/180.0)), 0>
  // вращение зеленой планеты
   }

  // красная планета
   sphere { <0, 2, 0> , 1
     texture {
          pigment { color Red }

          normal{bumps 0.9 scale 0.03}
```

```
        finish{ phong 1

        }
      }
    scale 0.2
    translate <-2*cos(360*clock*pi/180.0), -2-2*cos(360*(1-clock*pi/180.0)), 0>
 // вращение красной планеты
}
```



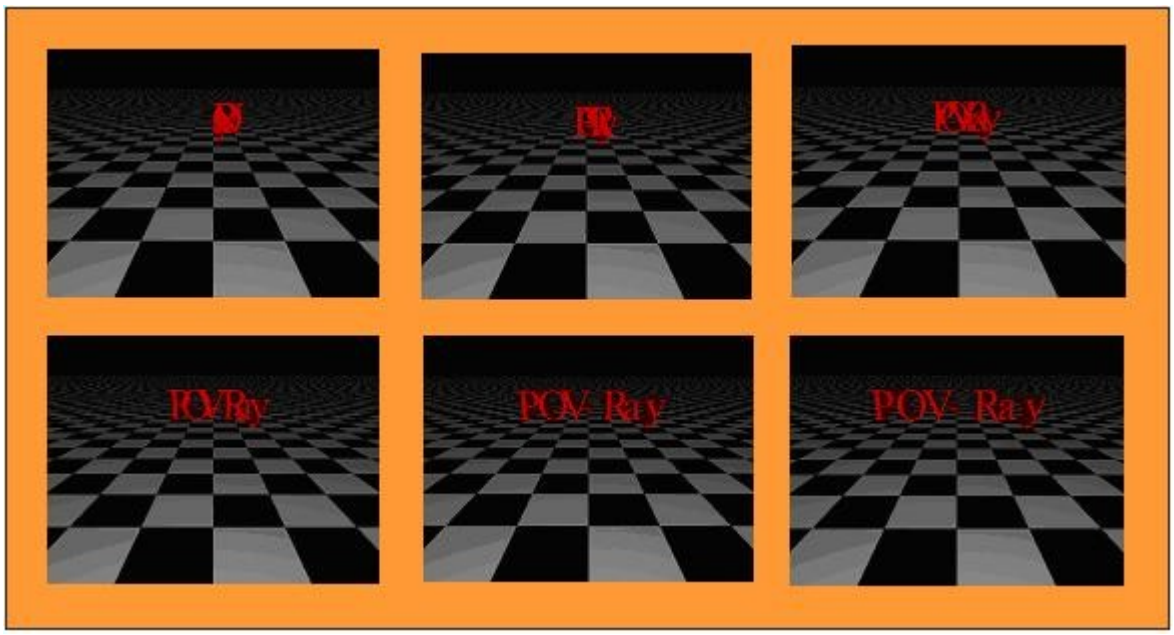
В нашем примере солнца вращается вокруг своей оси, с помощью командной строки rotate <0, clock*360, 0> . Вращение двух планет описано же более сложной функцией, используя свойства косинуса.

2.3. ДВИЖУЩИЙСЯ ТЕКСТ

Рассмотрим также текстовую анимацию. Сначала все буквы текста будут расположены в одной начальной точке, таким образом наслаиваясь друг на друга. И во время анимации они будут разъезжаться, таким образом красиво выстраиваясь в один ряд. Настройки файла QUICKRES.INI оставим прежними, изменим лишь основную часть кода.

```
# include "colors.inc"

camera {
      location < 0, 2, -6  >
      look_at < 0, 0, 0 >
}

light_source { < 0, 2, -6 > color White }

plane {
   y, 0
   pigment { checker color White color Black }
 }


text {
        ttf "timrom.ttf" "P"
          1, <0, 0, 0>
          pigment { Red }
          translate < -1 * clock, 1, 0 >
 }

text {
   ttf "timrom.ttf" "O"
          1, <0, 0, 0>
          pigment { Red }
          translate < -0.7 * clock, 1, 0 >
 }

text {
   ttf "timrom.ttf" "V"
          1, <0, 0, 0>
          pigment { Red }
          translate < -0.4 * clock, 1, 0 >
 }
```

```
text {
   ttf "timrom.ttf" "-"
         1, <0, 0, 0>
         pigment { Red }
         translate < -0.1 * clock, 1, 0>
  }

text {
   ttf "timrom.ttf" "R"
         1, <0, 0, 0>
         pigment { Red }
         translate <0.2 * clock, 1, 0 >
  }

text {
   ttf "timrom.ttf" "a"
         1, <0, 0, 0>
         pigment { Red }
         translate < 0.5 * clock, 1, 0 >
  }

text {
   ttf "timrom.ttf" "y"
         1, <0, 0, 0>
         pigment { Red }
         translate < 0.8 * clock, 1, 0 >
  }
```



Здесь мы для каждой буквы использовали одну и ту же функцию: translate < x * clock, 1, 0 > , изменяя лишь переменную x, чтобы буквы становились в ряд с одинаковыми промежутками. Код не сложный, в то же время с его помощью возможно создать ряд интересных мультипликационных картин.

III ПРЕОБРАЗОВАНИЕ POV-RAY ФРЕЙМОВ В AVI ФАЙЛ

При создании анимации в POV-Ray среде читатель скорее всего столкнется со следующей проблемой. После запуска анимации мы можем обнаружить в папке, куда мы сохраняем готовые анимации, множество bmp файлов. Количество bmp файлов напрямую зависит от числа фреймов, которые мы указываем в настройках файла QUICKRES.INI. И это может создать неудобство. Например, просмотреть созданную анимацию можно в том случае, если у нас установлена программа POV-Ray и добавлены необходимые линии в QUICKRES.INI файл. Но есть и более оптимальное решение. Такую информацию можно найти в различных форумах, где обсуждается данная тема. К сожалению, приложение Help не рассказывает об этой возможности.

Есть множество доступных программ, с помощью которых можно собрать ряд файлов-изображений в мультипликацию. Например, Bmp2avi, AVI Creator, AVIEDIT, VideoMach и многие другие. Эти программы очень просты в использовании, а эффект от анимации неповторим.

Avi Creator - http://www.bloodshed.net/avi.html
AVIEDIT - http://www.sumrallworks.com/freebies/buttonhole/multimedia/avi/editor.htm
Bmp2avi - http://willsoft.free.fr/
VideoMach - http://www.gromada.com/download.html

РЕЗЮМЕ

Pov-ray включает в себя поддержку большинство стандартных геометрический фигур, а также позволяет создавать свои собственные. Язык используемый в Pov-Ray схож по структуре и синтаксису с другими языками программирования, что позволяет очень быстро усвоить основные рабочие элементы скрипта.

Поддержка анимации происходит на том же скриптовом уровне, что описание фигур, что в свою очередь может включать любое кручение, вращение, перемещение, изменение цвета, текстур и так далее. Комбинация этих свойств позволяет создавать анимацию любой сложности.

Особое внимание можно уделить настройкам анимации при помощи QUICKRES.INI файла, где можно задать количество фреймов и временной промежуток анимации, при помощи данного функционала можно очень быстро замедлить или ускорить анимационных процесс.

Исходя из того, что анимация, например, вращения происходит из расчёта формул, где главная переменная это Время (этап анимации), то можно сделать вывод, что анимация в Pov-Ray больше подходит для демонстрации цикличной или технической анимации, как например вращения планет вокруг солнца, или вращение деталей технического изделия.

Целью данной работы было ознакомление читателя с возможностями сознания анимации в среде POV-Ray. Всего в данной работе рассмотрена 6 примера анимации, с помощью которых нам будет нетрудно понять схему создания анимации и в дальнейшем создать свой мультипликационный объект.

Данная работа не заменяет официальный источник информации о среде Pov – Ray, а является её дополнением. В этой работе содержится информация о том, как создать простые объекты анимации, а также является основой для начала практической части по созданию анимации.

KOKKUVÕTE

POV-Ray on vabavaraline 3D modelleerimisprogramm, mida kasutatakse fotorealistlike piltide renderdamiseks ja loomiseks. POV-Ray on kirjutatud SDL keeles ning toetab makrosid ja tsükleid. Programm sisaldab valmisobjektide-, tekstuuride- ja stseenide biblioteeke. Nendele baseerudes lihtsustub realistlike pilte ja animatsioonide loomine.
Antud töös on käsitletud POV-Ray animatsioonide loomise võimalust. Animatsioonide loomise probleemistik on POV-Ray juhendmaterjalides kajastatud mitte piisava põhjalikkusega. Selguse saamiseks tuli läbi viia suur hulk katsetusi.

Selgus, et animatsioonide loomisel on olulisteks mõisteteks muutuja “Clock”, samuti "Phase" ja need mõlemad on seotud failiga Quickres.ini. "Clock" ja "Phase" on kasutatavad dünaamilise parameetrina animatsiooni üksikute freimide genereerimisel. Nende abil saab moodustada tsükleid, sealhulgas mitme hierarhia tasemelisi. " Quickres.ini" failis määratakse ära freimide arv ja ajaline kestvus (initial clock ja final clock). Neid muutes saab mõjutada loodava animatsiooni kestvust, tempot ja kvaliteeti.

Omaette probleemiks osutus POV-Ray poolt genereeritud freimide transformatsioon animatsiooniks. POV-Ray foorumis oli antud suuniseid selle probleemi lahendamiseks, kuid sobivaima leidmine toimus katsetamise teel. Läbiviidud eksperimentide tulemusena soovitame kasutada vabavaralist programmi AviCreator ja VideoMach.

Käesoleva seminaritöö raames on välja pakutud animatsioonide loomise probleemile POV-Ray keskkonnas terviklik lahendus. Seda saab kasutada antud valdkonda sisseelamiseks ja edasi liikumiseks. Seminaritööle baseerudes saab asjast huvitatu hakata kohe liikuma õiges suunas. Meie poolt esitatud tüüpilisemate situatsioonide lahendused peaksid pakkuma piisavat tuge.

Tööle lisatud CD sisaldab seminaritöö teksti ja sellele lisaks näidisülesannete terviklahendusi (POV-Ray formaadis stseenide kirjeldus ja sellest saadud animatsioon avi formaadis).

ИСПОЛЬЗОВАННАЯ ЛИТЕРАТУРА

1. POV-Ray - Persistence Of Vision Ray Tracer: www.povray.org
2. Computer graphics: http://www.niac.ru/graphinfo.nsf/
3. Animations with POV-Ray: http://www.f-lohmueller.de/pov_tut/animate/pov_anie.htm
4. POV-Ray Help