Animation

- Animation overview
- Basics of animation in POV-Ray

Animation

- Why it is possible
- History
- Combining art and technology

Classical Animation

- Story
- Storyboard
- Soundtrack
- Detailed layout
- Layout <-->Sound
- Keyframes
- Inbetweening
- Pencil test
- Transfer to cels
- Paint cels
- Photograph cels

Twelve Techniques of Disney

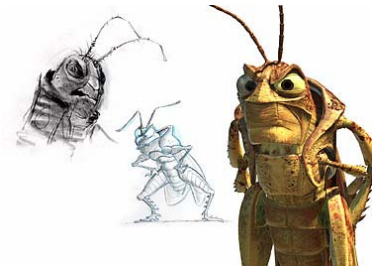
- Squash and Stretch
- Anticipation
- Staging
- Straight-ahead/pose-to-pose action
- Follow-through/overlapping action
- Slow-in/slow-out
- Arcs for motion
- Secondary action
- Timing
- Exaggeration
- Solid modeling
- Character personality

Quoted from Issac Kerlow.
The Art of 3D Computer Animation and Effects

Three techniques

Three techniques

Three Techniques



History of Animation

- Crossover of 3D animation with traditional animation
- *Who Framed Roger Rabbit?*
- *Luxo, Jr.* (http://www.pixar.com/shorts/ljr/)

Types of Animation Systems

- Low-level
- Procedural
- Representational
- Stochastic
- Behavioral

Low-level

- Scripting systems
- Keyframe systems
- Spline-driven

Procedural

- Movement as a function of time
- Visualize laws of physics
- “Cartoon Laws of Physics”
- POV-Ray example

Representational

- Allows an object to change shape
- Three categories:
  - Articulated objects *Luxo, Jr.*
  - Soft objects Cave Troll in *LOTR*
  - Morphing cat in *Harry Potter*

Stochastic Animation

- Controlled randomness
- Large groups of “actors”
- Examples:
  - Fireworks, fire, water falls
- Genesis sequence from *Star Trek II: The Wrath of Khan*

Behavioral Animation

- Rule-based
- Objects or “actors” react to their environment
- Examples
  - Schools of fish, flocks of birds
- *Stanley and Stella Break the Ice*
- Stampede scene from *The Lion King*
- Battle scenes in *LOTR*

POV-Ray

POV-Ray Animation

- POV Ray does NOT generate animations
- POV Ray generates the frames on separate .bmp files
- Frames are sequentially numbered in ascending order
- An external program to take those frames and put them into an animation is needed

POV-Ray animation

- There are two halves to animation support:
  - Telling POV Ray to render more than one frame
  - Modify the POV scene file to change on each frame

POV-Ray Animation

- To render more than one frame
  - Settings in the INI file (or on the command line)
- To change the scene on every frame
  - Clock and Phase keywords

INI Settings

- Two key things
  - Setting the range of frames to render
    - Initial_Frame
    - Final_Frame
  - Setting the time that occurs between the first and last frames
    - Initial_Clock
    - Final_Clock

INI Settings

- Example

```
Initial_Frame=1
Final_Frame=60
Initial_Clock=0
Final_Clock=1
```

  - POV Ray will render 60 frames. The clock will start at 0 and will end at 1, increasing at intervals of 1/60 for each frame.

INI Settings

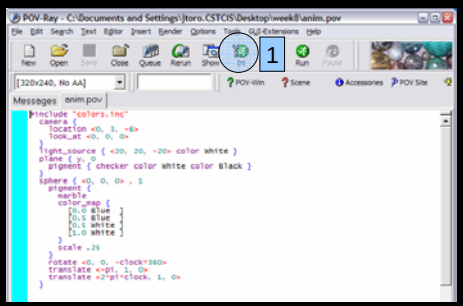
- You need to set this under the desired resolution entry in your INI file
- In the example here, if you select the `[320x240, 60F AA]` option, it will render 60 frames, but if you select the `[800x600, No AA]`, it will render one frame.

```
[800x600, No AA]
Width=800
Height=600
Antialias=On

[320x240, 60F AA]
Width=320
Height=240
Antialias=On
Initial_Frame=1
Final_Frame=60
Initial_Clock=0
Final_Clock=1
```

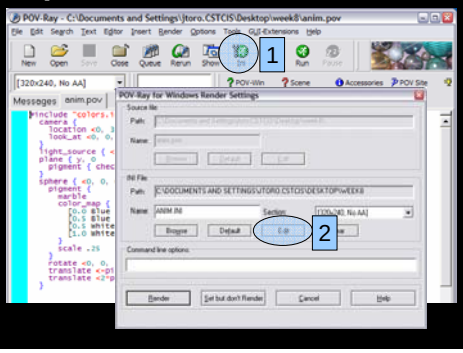
Modify INI file for animation

1. Locate the INI file
2. Open the INI file
3. Add the animation options
4. Select the animation options for rendering

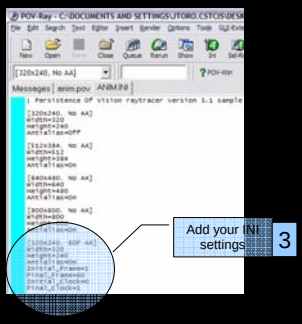
Step 1: Locate INI file



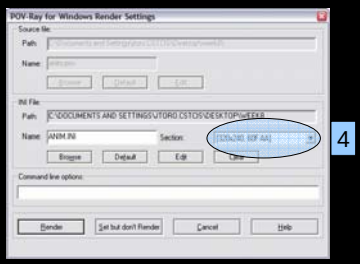
Step 2: Edit INI file



Step 3: Add animation settings



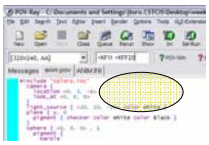
Step 4: Select the settings



Command Line settings

- Can set the animation values at the command line window

+KFIn Same as Initial_Frame=n
+KFFn Same as Final_Frame=n
+KIn.n Same as Initial_Clock=n.n
+KFn.n Same as Final_Clock=n.n



Code modifications

- The Clock variable
- Its value changes for each frame (automatically)
- By default, it goes from 0.0 to 1.0, no matter how many frames you have

POV-Ray

-
```
sphere {
   <0, 0, 0>, 1  + clock
}
```

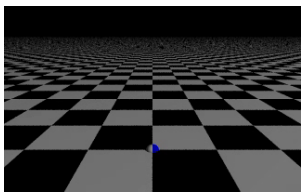
- ini file:

```
Initial_Frame = 1
Final_Frame = 20
Initial_Clock = 0.0
Final_Clock = 2.0
```

Growing Sphere

```
sphere { <0, 0, 0> , 1 + clock
    pigment {
      marble
      ...
    }
    translate <0, 1, 0>
}
```

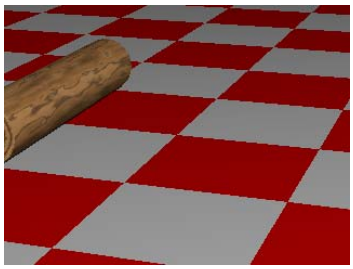
Result



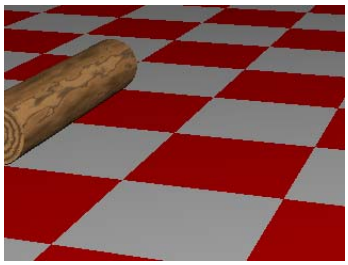
Movies from images

- pjBmp2Avi
  - Free simple program
  - Takes a sequence of images and dumps them into an AVI file

Rolling Log



Rolling Log and crash



Rolling Log and Crash

- Story board
  - 0-2.5 sec: roll right
  - 2.5 -3 sec: off stage
  - 3-6.5 sec: roll left
  - 7-8 sec: crash
- Statistics
  - 80 frames
  - 8 clock seconds

Rolling Log and Crash

```
  #if (clock <= 2.5)
         object {wlog
         rotate <0,0,clock*360>
         translate <0.21*2*pi*clock-1,0.21,0>

    }
  #else
```

Rolling Log and Crash

```
     #if (clock >= 3 & clock <= 6.5)
       #declare local_clock = clock - 3
       object {wlog
       rotate <0,0,clock*360>
       translate <3-0.21*2*pi*local_clock,0.21,0>}
  #end
#end
```

Rolling Log and Crash

```
#if (clock <7)
   camera {
        location  <2, 2, -3>
        look_at <0, 0, 3>
}
```

Rolling Log and Crash

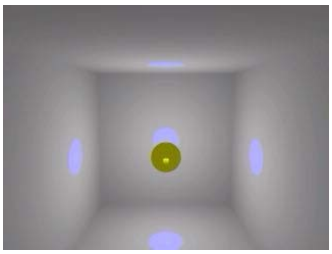
```
#else
      #declare pos = seed (8723*clock);
      #declare py = rand(pos);
      #declare px = rand(pos);
      camera {
        location  <2, 2, -3>
        look_at <((px-0.5)*0.8), (py-0.5)*0.8, 3>
       }
#end
```

Ceiling Fan

Animating light

```
#declare px = clock*12;
  light_source {
     <20,20,-20>
     color White
     spotlight
     radius 1.5
     falloff 2.5
     point_at <px-6,0,0>
  }
```

Animating light



Animating light

```
#declare GlitterBall = difference {
       sphere {<0,0,0>, 1}
       cylinder {<0,-1,0> <0,1,0> 0.2}
       cylinder {<-1,0,0> <1,0,0> 0.2}
       cylinder {<0,0,-1> <0,0,1> 0.2}
 }
```

Animating light

```
#declare a = seed(clock*1000);
#declare b = seed(clock*200);
#declare cr = rand(a);
#declare cg = rand(b);

light_source {
   <0, 0, 0>
   color rgb <cr,cg,1>
   fade_distance 0.5 }
```

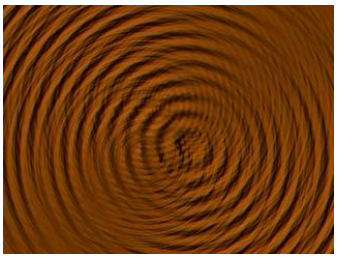
Animating light

```
object { GlitterBall
   pigment {color Yellow}
   finish {ambient 0.5}

   rotate <-clock*360, 0, -clock*360>
   scale 0.5

 }
```

Animating textures

```
texture {
       pigment {color Orange}
       normal{
            waves 0.8*(1-clock)
            scale 0.3
            frequency 15*(1-clock/2)
       }
  }
```

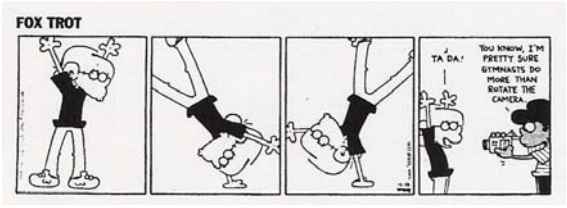
Animating textures



Animating the camera

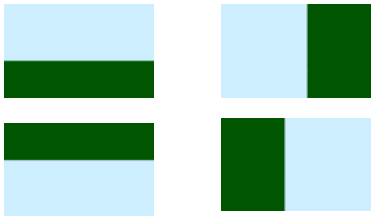
- Sky keyword
- You can also rotate/translate the camera

Sky



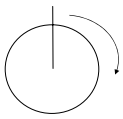
Sky vector

- "View up"



Animate Sky vector

```
#include "functions.inc"

sky <cos(pi/2 + 2*pi*clock),
     sin(pi/2 + 2*pi*clock),
     0>
```



Animating camera

Vertigo

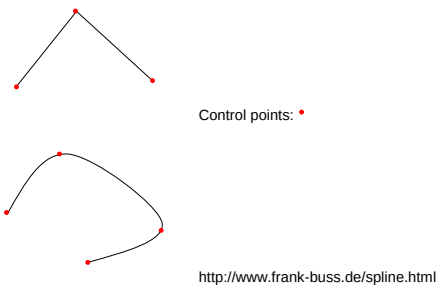
Animating the camera

- Define a path to follow
- Splines give you a way to define 'pathways'

Types of Splines

- Polygonal arcs (linear spline)
- Cardinal splines
- B-splines
- Bezier curves
- Nurbs (non-uniform rational b-splines)

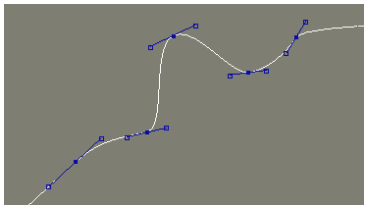
Arcs and Cardinal Splines



http://www.frank-buss.de/spline.html

Bezier curves

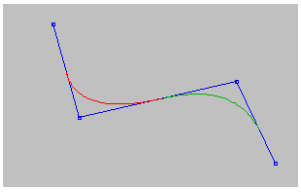
•shape defined by control points and
•tangents



http://webreference.com/3d/lesson36/part2.html

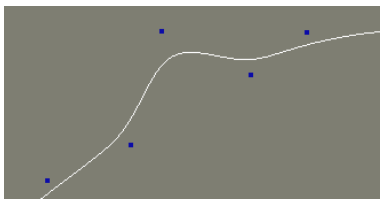
B-splines

•control points not necessarily on spline
•control points shape spline



http://www.sunsite.ubc.ca/LivingMathematics/V001N01/UBCExamples/Bezier/bezier.html
http://www.ibiblio.org/e-notes/Splines/Basis.htm

B-splines and Nurbs



Nurbs are B-splines that allow weighting of control points

http://webreference.com/3d/lesson36/part2.html

Degree of a spline

- higher degree creates smoother spline
- needs more control points

http://i33www.ira.uka.de/applets/mocca/html/noplugin/curves.html

Spline

```
#declare IDENTIFIER =
   spline {
     [SPLINE_IDENTIFIER] |
     [SPLINE_TYPE] |
     [Val1, <Point1>[,]
      Val2, <Point2>[,] ...
      Valn, <Pointn>]
   }
SPLINE_TYPE: linear_spline | quadratic_spline | cubic_spline |
   natural_spline
SPLINE_USAGE: IDENTIFIER(Val) | IDENTIFIER(Val, SPLINE_TYPE)
```

Spline Example

```
#declare MySpline =
  spline {
    cubic_spline
    -.25, <0,0,-1>
    0.00, <1,0,0>
    0.25, <0,0,1>
    0.50, <-1,0,0>
    0.75, <0,0,-1>
    1.00, <1,0,0>
    1.25, <0,0,1>
  }
```

A cubic spline is declared

Points defining the curve

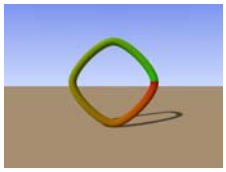
Spline Example

```
#declare ctr = 0;
#while (ctr < 1)
  sphere {
    MySpline(ctr),.25
    pigment { rgb <1-ctr,ctr,0> }
    rotate <90,0,0>
    translate <0,1.1,0>
  }
  #declare ctr = ctr + 0.01;
#end
```

The spline is used to define the location of the spheres

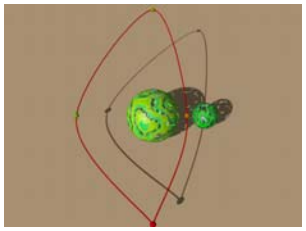
Spline Example

```
#declare ctr = 0;
#while (ctr < 1)
  sphere {
    MySpline(ctr),.25
    pigment { rgb <1-ctr,ctr,0> }
    rotate <90,0,0>
    translate <0,1.1,0>
  }
  #declare ctr = ctr + 0.01;
#end
```



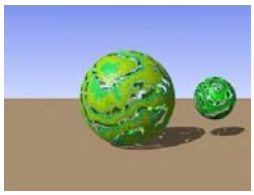
Camera following a Spline

```
#declare MySpline =
  spline {
    cubic_spline
    -0.25, <0,1,-4>
    0.00, <0,1,-4>
    0.25, <1.3,1,0>
    0.50, <0,3,4.5>
    0.75, <-3,1,0>
    1.00, <0,1,-4>
    1.25, <0,1,-4>
  }
```



Camera following a Spline

```
camera {
    location MySpline(clock)
    look_at   <0.0 , 1.0 , 0.0>
}
```



Boom

- sky <0,0,1> and
  - distance <0,0,1> or
  - look_at – location is parallel to sky

Phase

- For textures, especially those that can take a color, pigment, normal or texture map. Remember the form that these maps take:

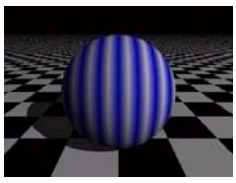
```
color_map {
   [0.00 White ]
   [0.25 Blue ]
   [0.76 Green ]
   [1.00 Red ]
}
```

Phase

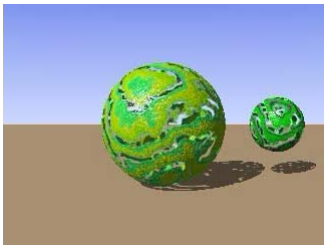
- Phase causes the color values to become shifted along the map by the amount specified in phase.
- If clock value is from 0.0 to 1.0, use it with phase, and the pattern will smoothly shift over the course of the animation.

Phase example

```
sphere { <0, 0, 0> , 1
    pigment {
      marble
      color_map {
        [0.0 Blue  ]
        [0.5 Blue  ]
        [0.5 White ]
        [1.0 White ]
      }
      phase clock
      scale .25
    }
    translate <0, 1, 0>
  }
```



Phase example



Selecting frames to render

- Good for long animations
- Setting
  - Initial_Frame=n and
  - Final_Frame=m won't work.
- Use
  - Subset_Start_Frame=n
  - Subset_End_Frame=m