Lights and Cameras in Renderman

Using the Python API

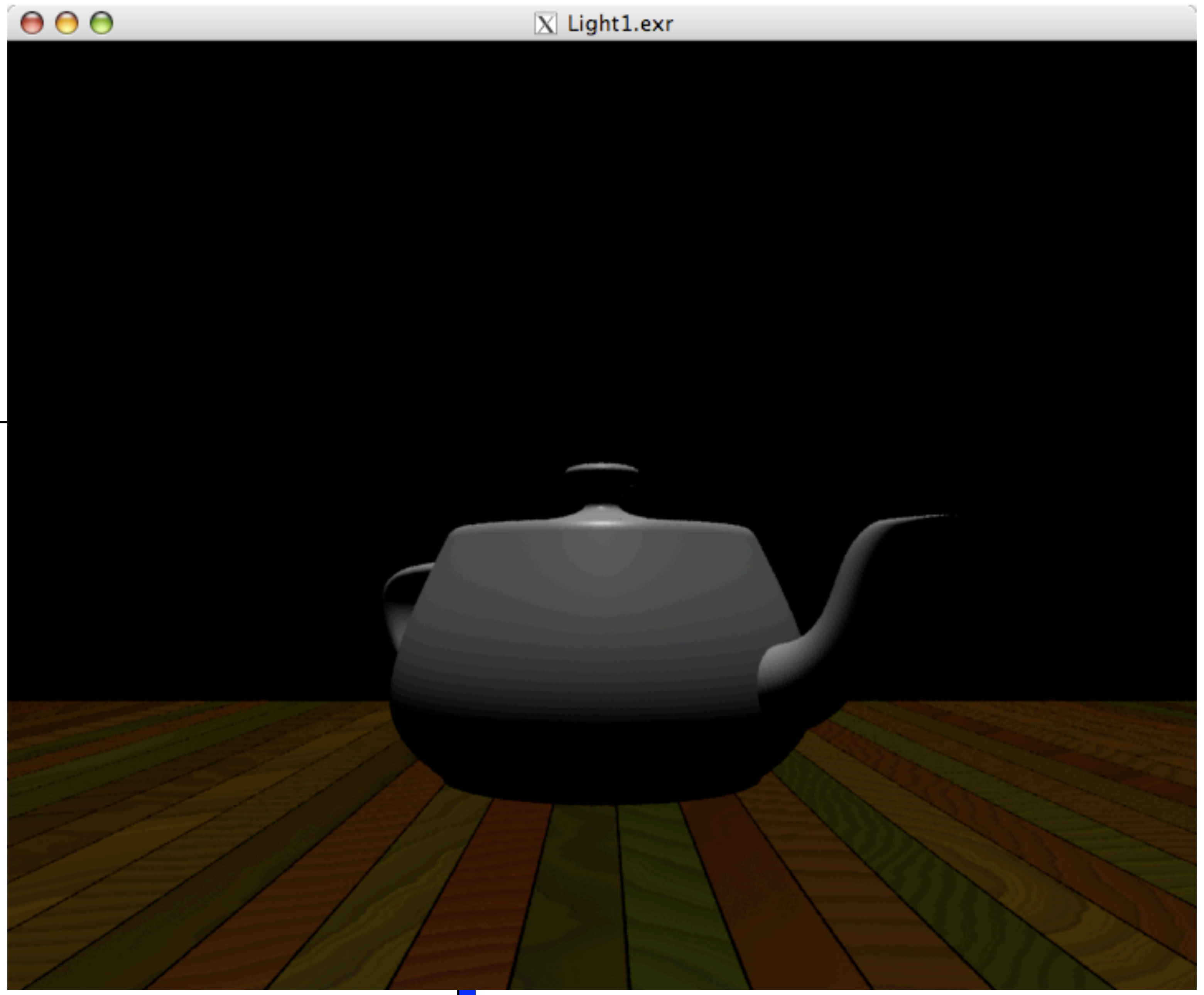
Lighting

- All the scenes developed so far have used a simple lighting model which is part of the default surface
- To use Real lights in Renderman we need to add a surface to the Objects to be rendered.
- To do this we use the Surface command as follows

```
ri.Surface("plastic")

Surface "plastic"
```

- Using this with no lights the scene will appear black

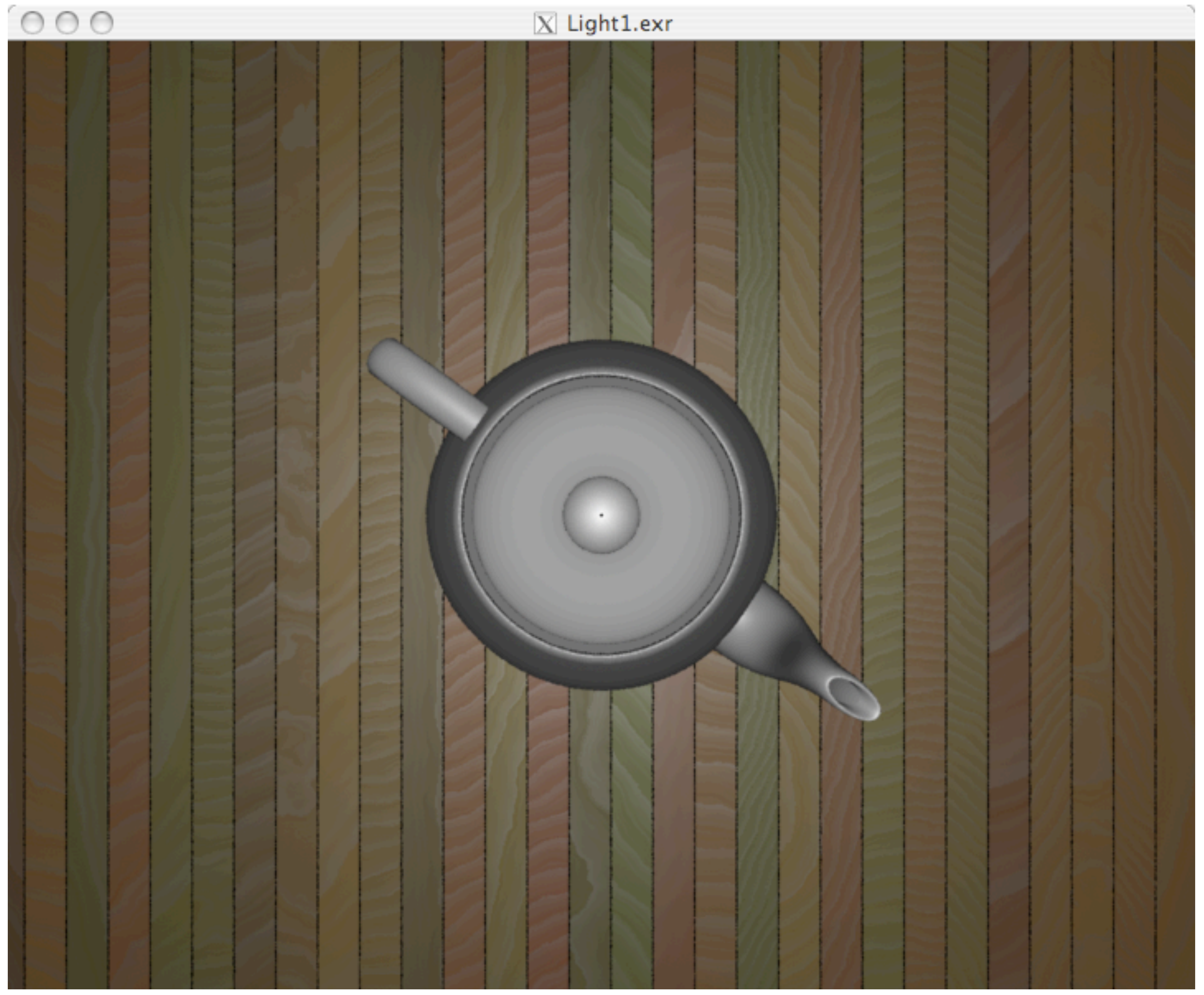
Basic Scene



```python
def Scene(ri) :
  ri.AttributeBegin()

  face=[-0.1,-1,-3, 0.1,-1,-3,-0.1,-1,3, 0.1,-1,3]
  random.seed(25)
  plank=-5.0
  while (plank <=5.0) :
    ri.TransformBegin()
    ri.Color([random.uniform(0.2,0.4),random.uniform(0.1,0.25),0])

    c0=[random.uniform(-10,10),random.uniform(-10,10),random.uniform(-10,10)]
    c1=[random.uniform(-10,10),random.uniform(-10,10),random.uniform(-10,10)]
    ri.Surface("wood",{"point␣c0":c0,"point␣c1":c1,"float␣grain":random.randint(2,20)})
    ri.Translate(plank,0,0)
    ri.Patch("bilinear",{'P':face})
    ri.TransformEnd()
    plank=plank+0.206
  ri.AttributeEnd()
  ri.TransformBegin()
  ri.AttributeBegin()
  ri.Color([1,1,1])
  ri.Translate( 0,-1.0,0)
  ri.Rotate(-90,1,0,0)
  ri.Rotate(36,0,0,1)
  ri.Scale(0.4,0.4,0.4)
  ri.Surface("plastic")
  ri.Geometry("teapot")
  ri.AttributeEnd()
  ri.TransformEnd()
```

Scene

- The scene uses the random functions from python to change shader parameters for each plank
- The colour is also changed to add visual interest
- We use the seed function with a set value else we would get a different image each time we render
- As you will see later in the year we could write our own shader to do this but for now we will do it with python



Renderman Shaders

- Shaders come in different forms but are basically in the following categories
  1. Surface
  2. Displacement
  3. Lighting
  4. Volume

Shaders 2

- The standard shaders in prman live in the following directory

```
/opt/pixar/RenderManProServer-14.0/lib/shaders
```

- The standard ones are as follows

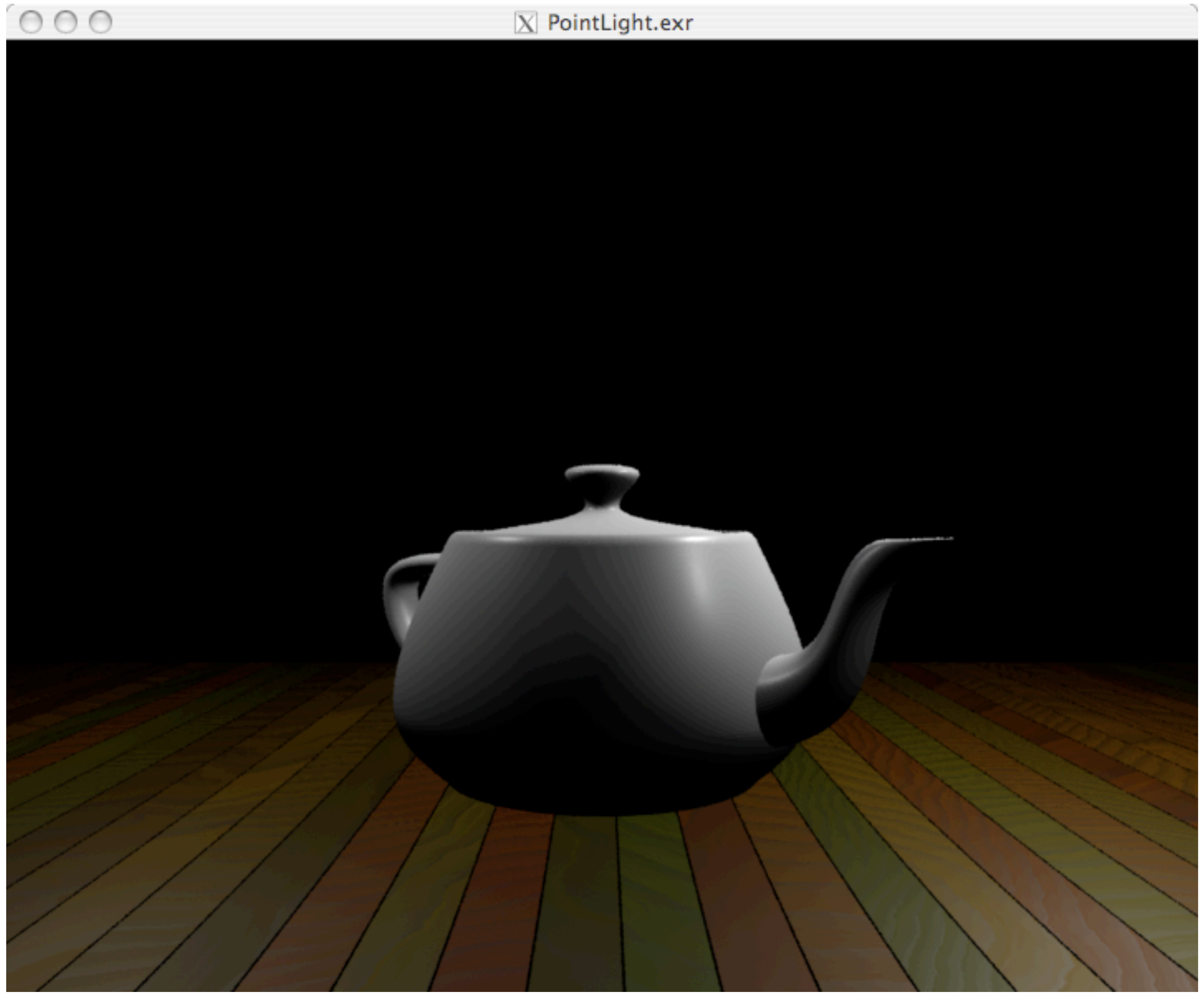
```
aachrome aaglass ambientlight bumpy carpet causticlight cloth cmarble
constant cosinelight_rts cylinderlight defaultlight defaultsurface
depthcue diaknurl disklight distantlight fog glassbal glassrefr glass
indirectlight indirectsurf linearlight maps matte metal null
paintedplastic plastic pointlight_rts pointlight pointnofalloff
rectanglelight rmarble rsmetal  shadowdistant shadowpoint shadowspot
shinymetal sinknurl spatter spherelight  spotlight stippled stone
texmap threads wood
```

Lights and Transformations

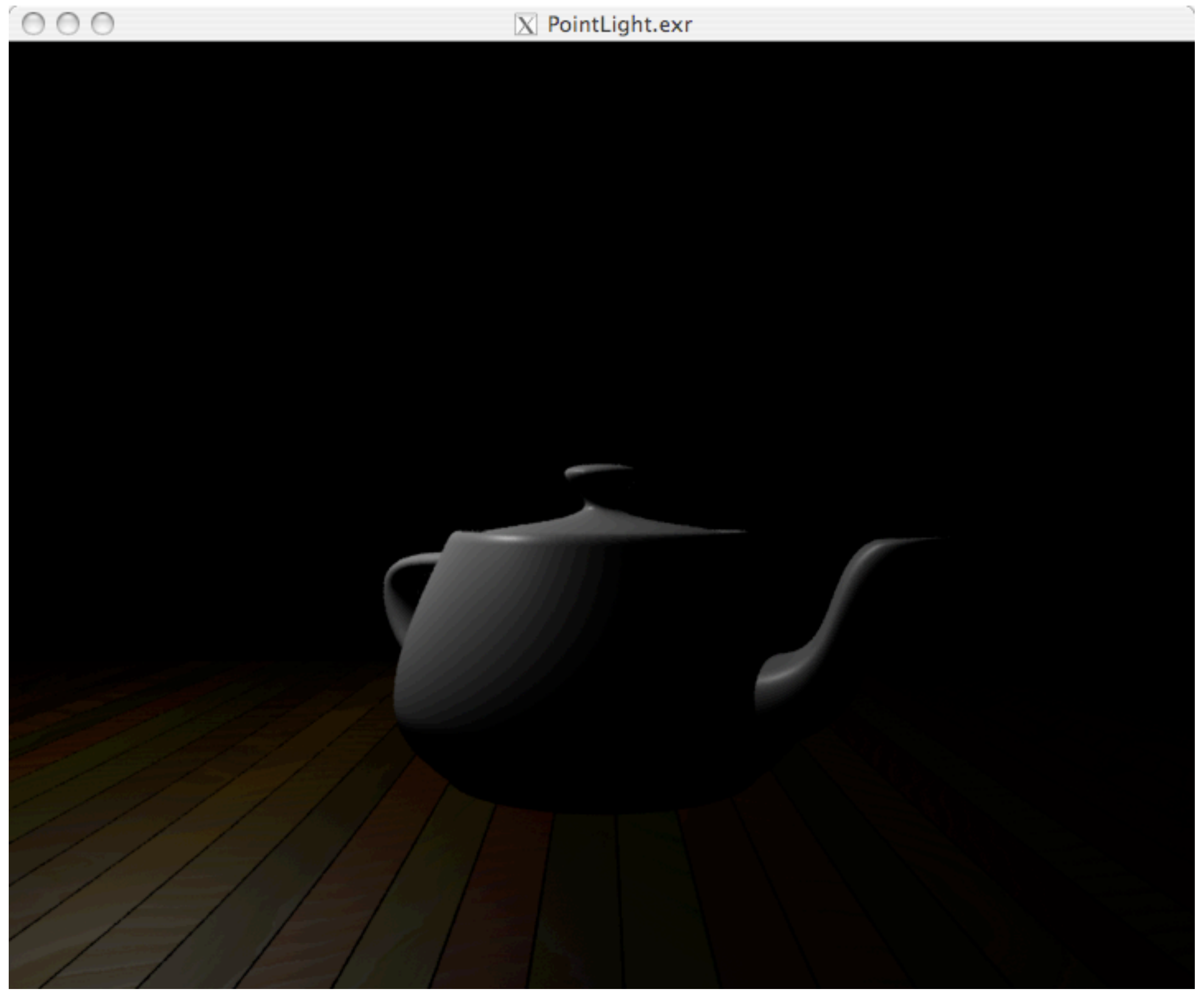
- Most lights pass in some form of point value which specifies a position.
- These values will be modified by the current transformation matrix so care must be used when specifying lights as to where they are to be positioned

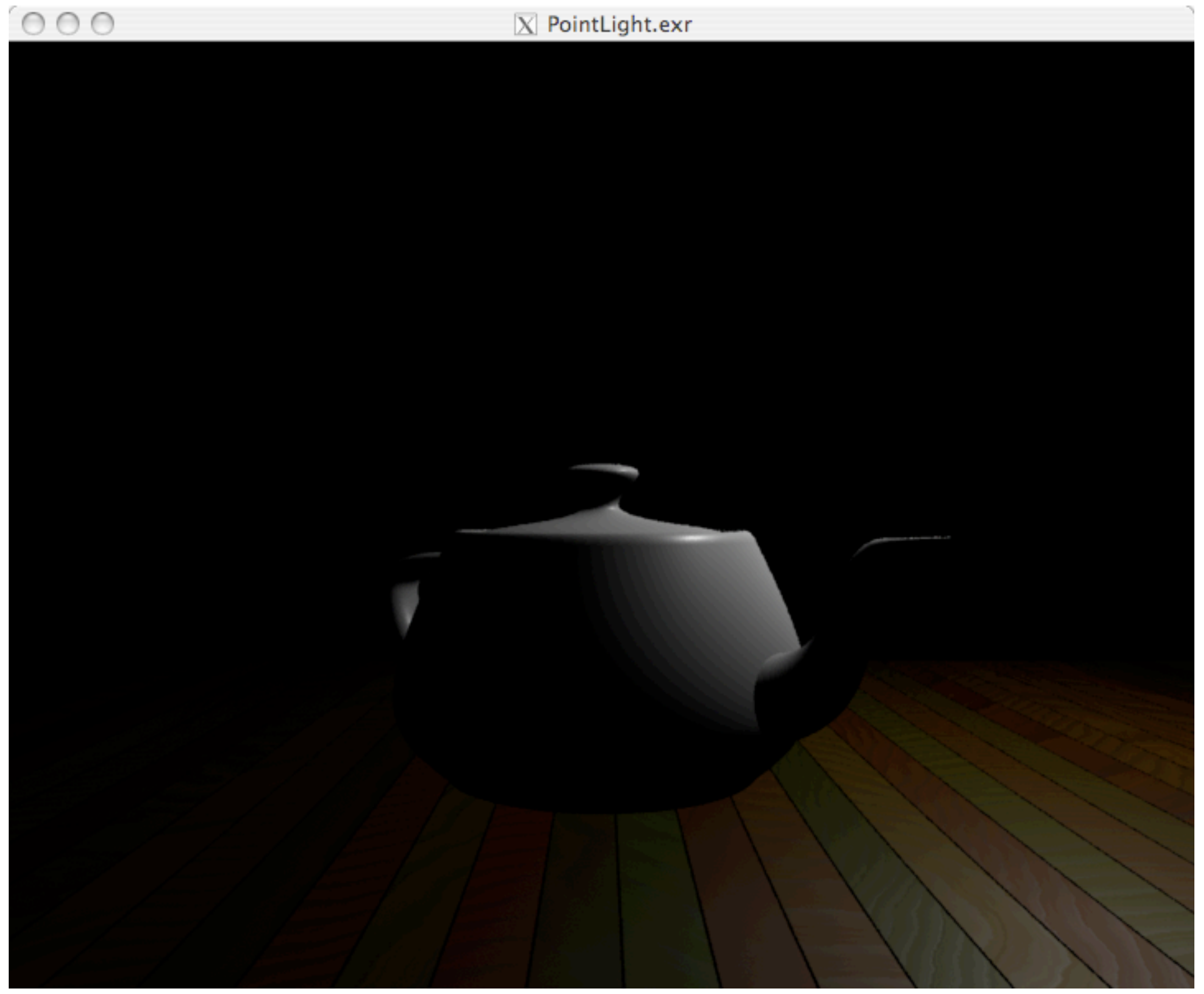
PointLight

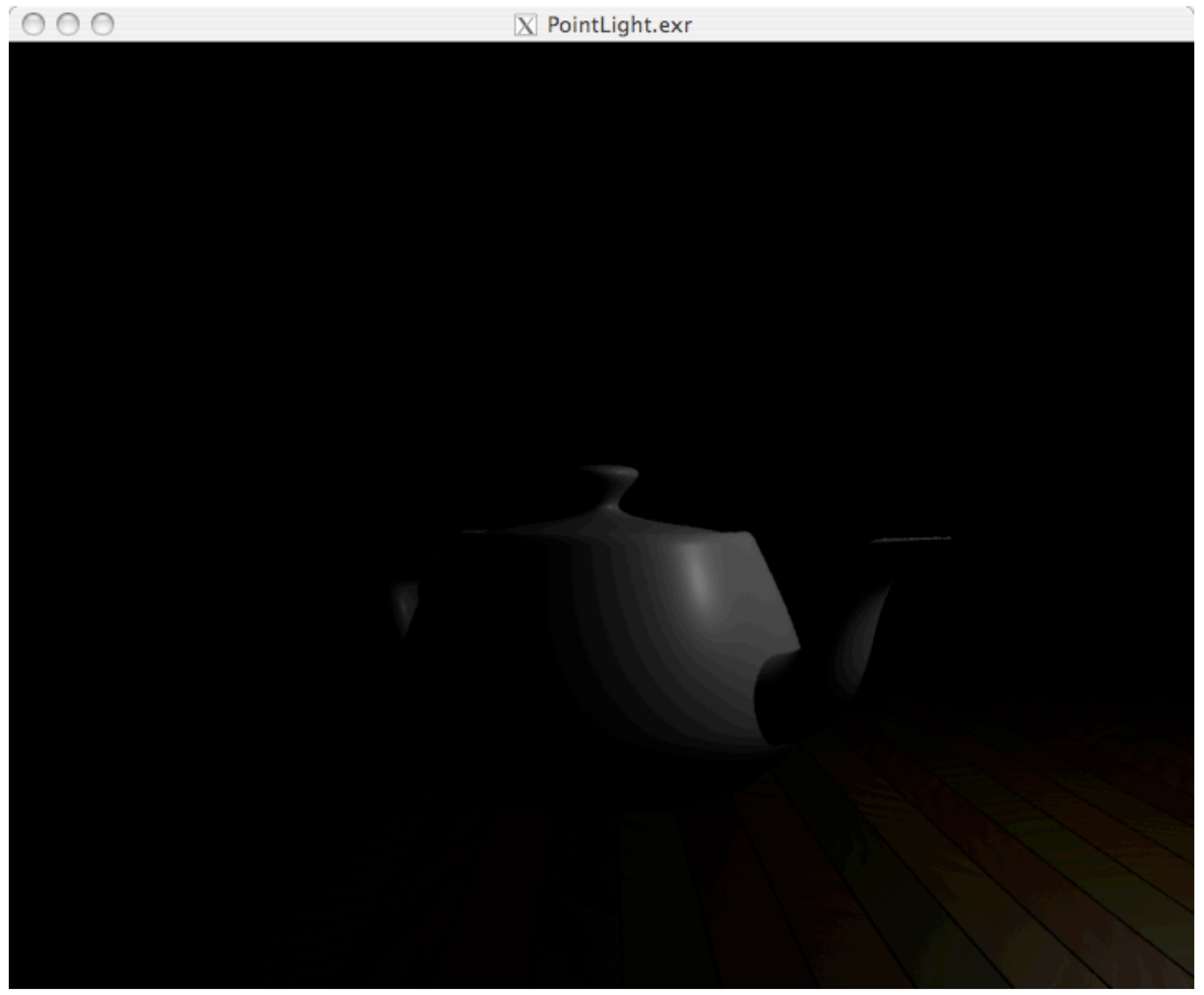
- Lights are created in a rib file using the LightSource command
- These control shaders to produce the lighting calculations for the scene
- The simplest light source is the PointLight which creates a light that shines equally in all directions
- Note that these lights are affected by the transformations
- The following file shows this in action



```
ri = prman.Ri() # create an instance of the RenderMan interface
ri.Option("rib", {"string␣asciistyle": "indented"})

filename = "PointLight.rib"
# this is the begining of the rib archive generation we can only
# make RI calls after this function else we get a core dump
ri.Begin(filename)

ri.Declare("Light1" ,"string")
ri.Declare("Light2" ,"string")
ri.Declare("Light3" ,"string")

# now we add the display element using the usual elements
# FILENAME DISPLAY Type Output format
ri.Display("PointLight.exr", "framebuffer", "rgba")
# Specify PAL resolution 1:1 pixel Aspect ratio
ri.Format(720,575,1)
# now set the projection to perspective
ri.Projection(ri.PERSPECTIVE,{ri.FOV:50})
# now we start our world
ri.WorldBegin()


ri.LightSource( "pointlight", {ri.HANDLEID:"Light1", "point␣from":[-2,2,4], "float␣
    intensity": [6]})

ri.TransformBegin()
ri.Translate(2,2,4)
ri.LightSource("pointlight", {ri.HANDLEID:"Light2", "point␣from":[0,0,0] ,"float␣
    intensity" :[8]})
ri.TransformEnd()

ri.TransformBegin()
ri.LightSource("pointlight",{ri.HANDLEID: "Light3","point␣from": [2,1,3] ,"float␣
    intensity": [2]})

ri.Illuminate("Light1",1)
ri.Illuminate("Light2",1)
ri.Illuminate("Light3",1)

ri.Translate(0,0,4)
Scene(ri)
ri.TransformEnd();

ri.WorldEnd()
ri.End()
```







```
Declare "Light1" "string"
Declare "Light2" "string"
Declare "Light3" "string"
Display "PointLight.exr" "framebuffer" "rgba"
Format 720 575 1
Projection "perspective" "uniform␣float␣fov" [50]
WorldBegin
  LightSource "pointlight" "Light1" "point␣from" [-2 2 4] "float␣intensity" [6]
  TransformBegin
    Translate 2 2 4
    LightSource "pointlight" "Light2" "point␣from" [0 0 0] "float␣intensity" [8]
  TransformEnd
  TransformBegin
    LightSource "pointlight" "Light3" "point␣from" [2 1 3] "float␣intensity" [2]
    Illuminate "Light1" 1
    Illuminate "Light2" 1
    Illuminate "Light3" 1

    .....................
```

The images above show the individual lights operated by Illuminate

Turning Lights on and Off

- In the previous example the light is given a numeric id via the Declare function
- This value can be used in conjunction with the Illuminate function to turn the light on and off
- The function uses the following format

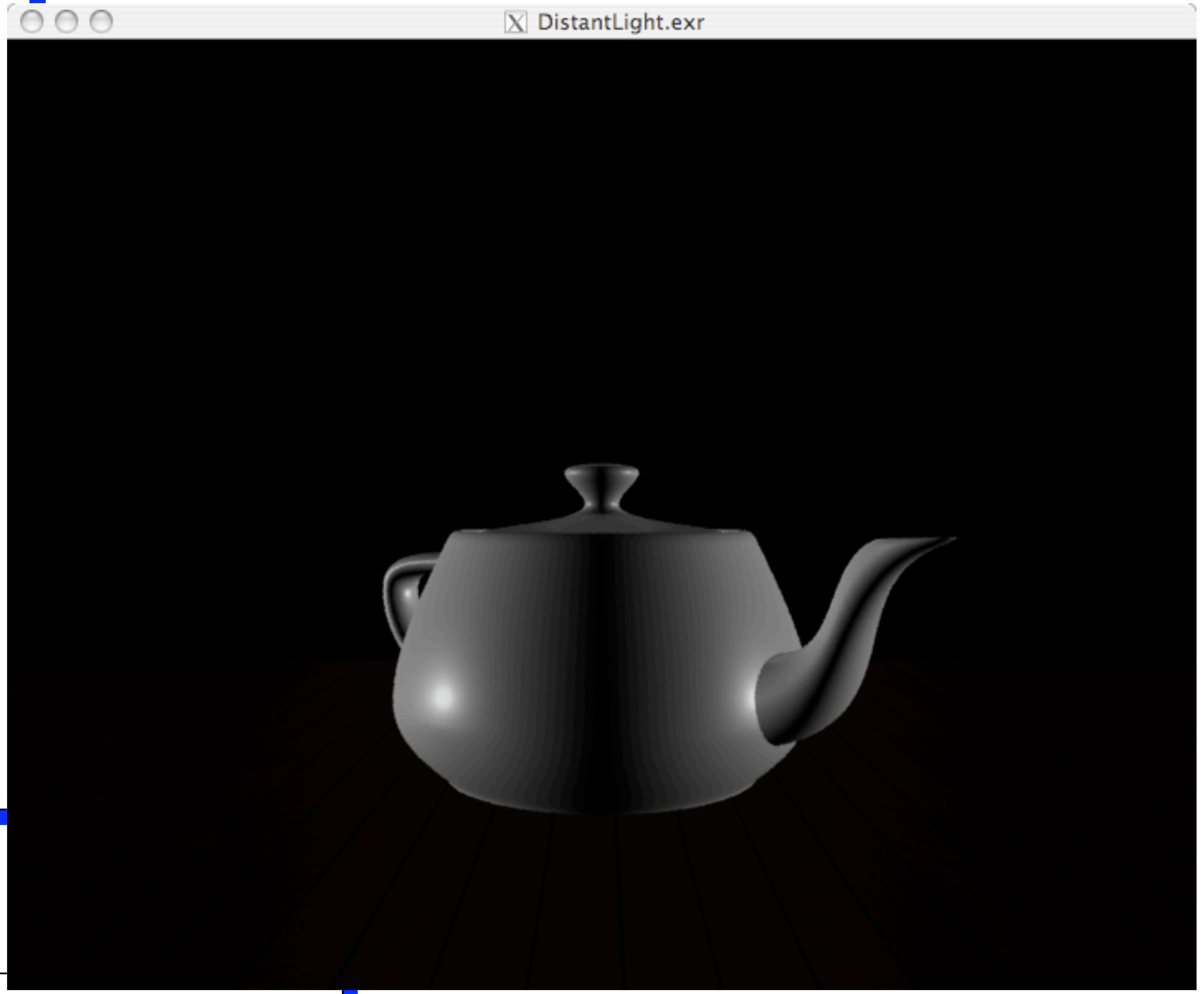
```
Illuminate id [0 off 1 on]
```

- As the state of the light is an attribute it is possible to save and restore the values using the AtributeBegin / End blocks

DistantLights

- Distant lights are typically used to represent daylight
- They have orientation but no position, and this position is specified by the "to" parameter as shown in the following example

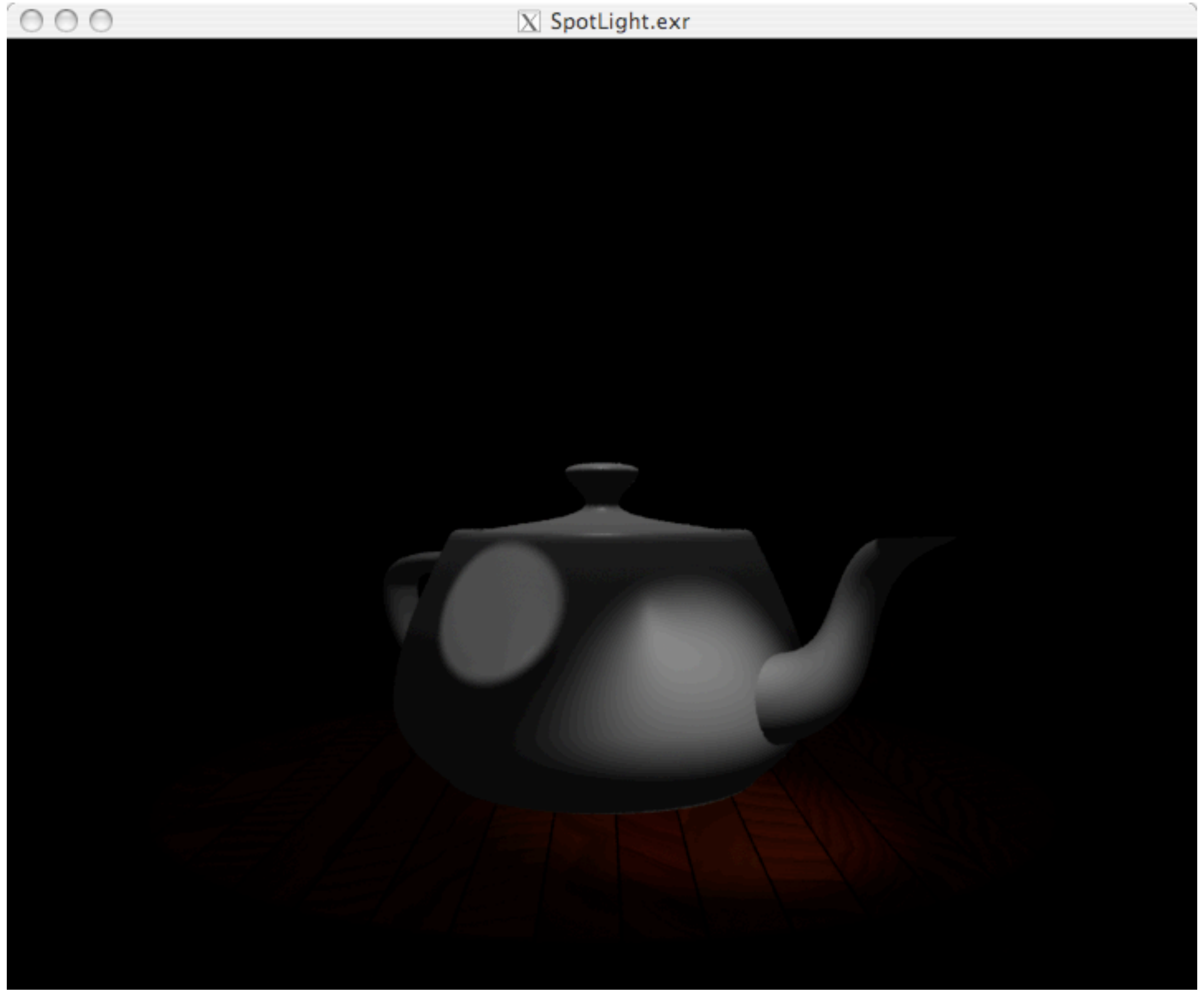
```python
ri.WorldBegin()


ri.LightSource( "distantlight", {ri.HANDLEID:"Light1", "point_to":[1,-0.03,0],
      "float_intensity": [1]})
ri.LightSource( "distantlight", {ri.HANDLEID:"Light2", "point_to":[-1,-0.03,0]
     , "float_intensity": [1]})
ri.LightSource( "distantlight", {ri.HANDLEID:"Light3", "point_to":[0,-0.5,-1],
      "float_intensity": [0.2]})

ri.Illuminate("Light1",1)
ri.Illuminate("Light2",1)
ri.Illuminate("Light3",1)



ri.Translate(0,0,4)
Scene(ri)


ri.WorldEnd()
# and finally end the rib file
ri.End()
```



```
WorldBegin
  LightSource "distantlight" "Light1" "float_intensity" [1] "point_to" [1 -0.03 0]
  LightSource "distantlight" "Light2" "float_intensity" [1] "point_to" [-1 -0.03 0]
  LightSource "distantlight" "Light3" "float_intensity" [0.2] "point_to" [0 -0.5 -1]
  Illuminate "Light1" 1
  Illuminate "Light2" 1
  Illuminate "Light3" 1
```

SpotLights

- Spotlights give more control over the lighting of a scene by allowing the user to specify the cone size of the light and the position the light is pointing to.
- This is shown in the following example



```
#spot directly above
ri.LightSource( "spotlight", {ri.HANDLEID:"Light1",
        "from" : [0,2,4],
              "to" :[0,0,4 ],
              "intensity" : [2],
              "coneangle" : [0.5],
              "conedeltaangle" : [0.01]})

ri.TransformBegin()
ri.Translate(0,0,4)
ri.LightSource( "spotlight",{ri.HANDLEID: "Light2",
              "from" : [1,1 ,-2],
              "to" :[0, -2 ,0],
              "intensity": [4],
              "coneangle" :[0.3],
              "conedeltaangle" :[0.2],
              "lightcolor" :[1, 1, 1]})
ri.TransformEnd()

ri.LightSource ("ambientlight",{ri.HANDLEID: "Ambient","intensity" :[0.05]})

ri.Translate(0,0,4)
ri.LightSource( "spotlight",{ri.HANDLEID : "Light3",
        "from" : [-3,1,-5],
        "to" :[0, -0.5, 0],
        "intensity" :[10],
        "coneangle": [0.05],
        "conedeltaangle" :[0.01]})
```

Spotlight Rib Code

```
WorldBegin
  LightSource "spotlight" "Light1" "from" [0 2 4] "conedeltaangle" [0.01]
        "coneangle" [0.5] "to" [0 0 4] "intensity" [2]
  TransformBegin
    Translate 0 0 4
    LightSource "spotlight" "Light2" "lightcolor" [1 1 1] "from" [1 1 -2] "conedeltaangle" [0
        .2]
          "coneangle" [0.3] "to" [0 -2 0] "intensity" [4]
  TransformEnd
  LightSource "ambientlight" "Ambient" "intensity" [0.05]
  Translate 0 0 4
  LightSource "spotlight" "Light3" "from" [-3 1 -5] "conedeltaangle" [0.01]
        "coneangle" [0.05] "to" [0 -0.5 0] "intensity" [10]
  Illuminate "Light1" 1
  Illuminate "Light2" 1
  Illuminate "Light3" 1
  Illuminate "Ambient" 1

```

SpotLight Parameters

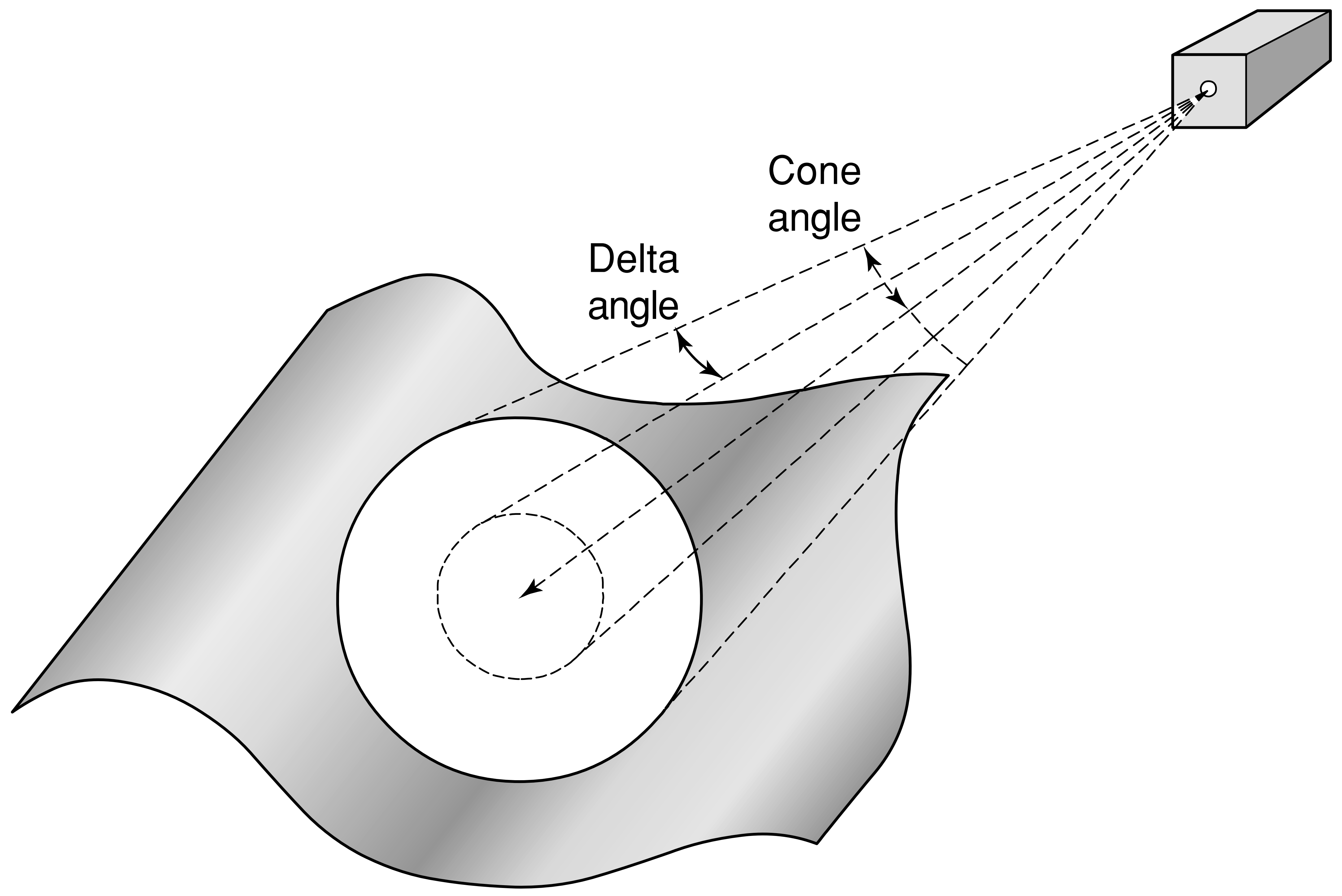


Figure 11.5. Cone angle and delta angle

Image from Esential Renderman Fast. Stephenson 2003. Springer.

Ambient Light

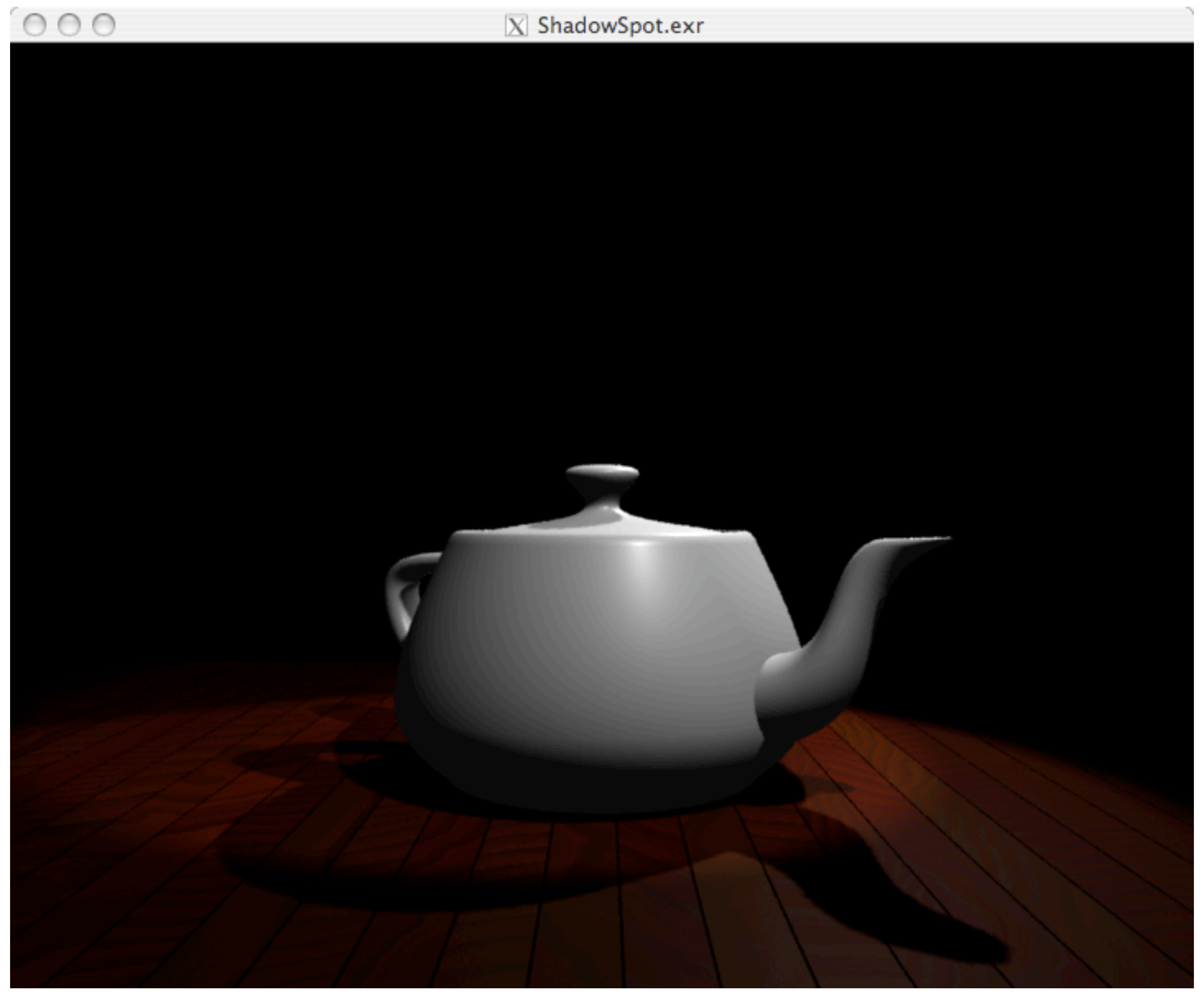
- Ambient Light is used to add a global light to the scene which illuminates all objects.
- It should not be set too high as it will make the scene washed out.

LightColour

- All lights are by default white, their colour may be changed using the lightcolor parameter

```

ri.LightSource( "spotlight",{ri.HANDLEID: "Light2",
                "from" : [1,1 ,-2],
                "to" :[0, -2 ,0],
                "intensity": [4],
                "coneangle" :[0.3],
                "conedeltaangle" :[0.2],
                "lightcolor" :[1, 1, 1]})

LightSource "spotlight" "Light2" "lightcolor" [1 1 1] "from" [1 1 -2] "conedeltaangle" [0.2]
       "coneangle" [0.3] "to" [0 -2 0] "intensity" [4]

```

Shadows



- Adding shadows is one of the most effective ways of adding realism to a computer-generated image.
- The human visual system uses shadows to determine depth, light location, and direction, as well as spatial relationships between objects.
- PhotoRealistic RenderMan supports shadows using a shadow map algorithm
- It can also produce shadows using standard ray tracing

Renderman Shadows

- The basic shadow map method is relatively easy to use, but it is not automatic.
- The basic process is simple.
  - For each light that casts a shadow a shadow map image must be rendered.
  - This image is rendered from the location of the light source.
  - The image is then converted into a format suitable for use by special light shaders which can use the map to selectively light the scene.
- This selective lighting produces shadows.

Shadow Maps

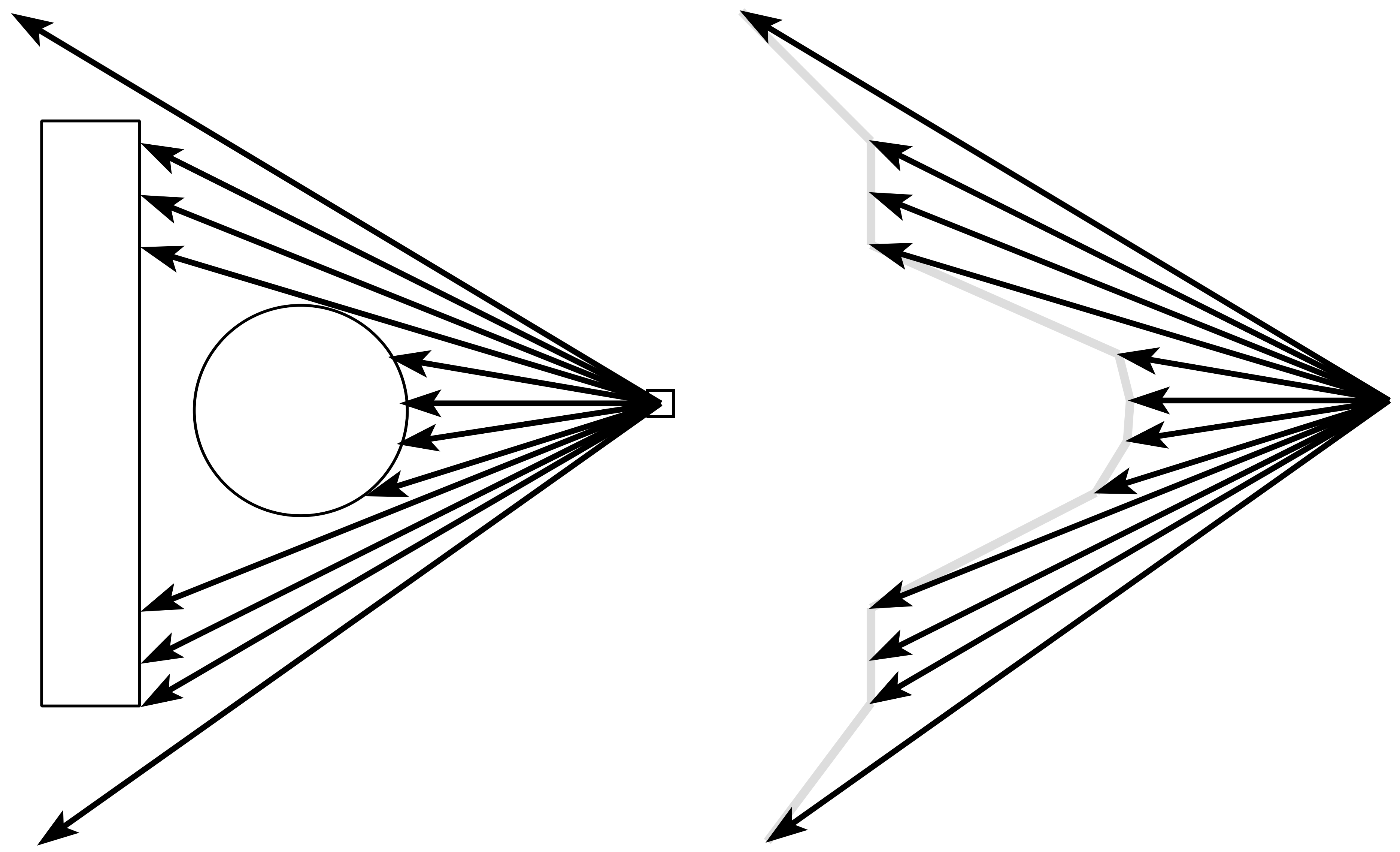


Image from Essential Renderman Fast. Stephenson 2003. Springer.

- The shadow map is different from a normal render as it only records the depth of the object in the image
- This can then be used to calculate the shadows from the point of view of the light.

Lighting Types

- Depending upon the type of light we may have to calculate many shadow maps
- For a spotlight one shadow map will suffice as it it a directional light and only throws light from one direction
- PointLights have an effective solid illumination angle of 360° we can't make shadow maps to cover the complete solid angle
- Therefore maps are made for all six major directions in world space +x -x, +y, -y +z -z.
- To avoid artifacts at the edges of the shadow maps they should be made with a field of view larger than 90° usually 95° is sufficient

FOV and Shadows

- Lights that are located at a point, such as spot lights and point lights, should have their shadow maps rendered using a perspective projection.
- The field of view for the perspective projection for a spot light should be calculated from the light cone angle.
- The cone angle for the standard RenderMan shaders is half the angle of the illuminated cone and is measured in radians, but the perspective field of view is measured in degrees and is the complete angle.
- The formula to convert from a cone angle to a field of view (fov) is:

```
fov = coneangle*360/pi
```

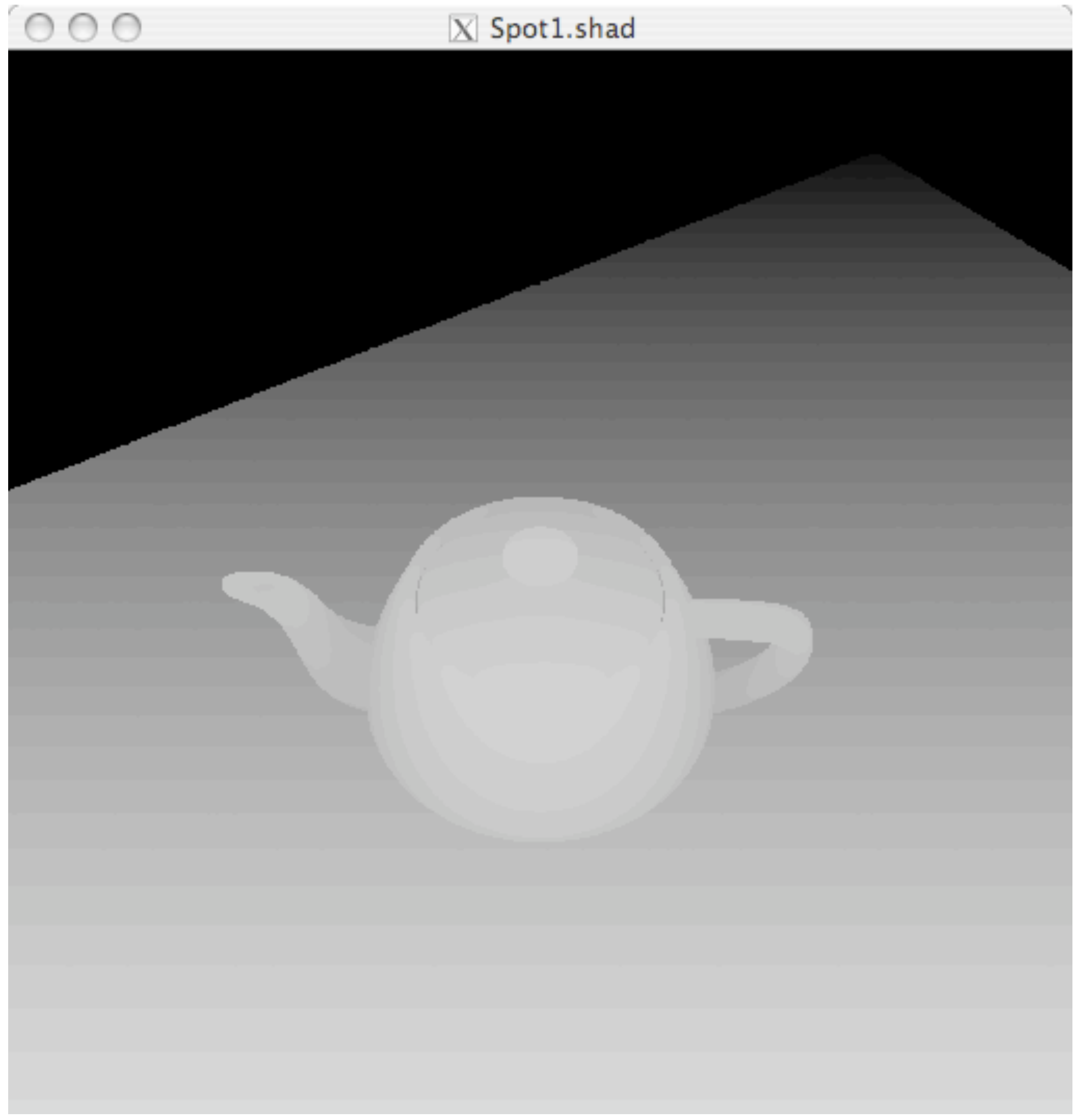
Creating a Shadowmap

- To create a shadow map we must render to the z buffer from the position of the light
- This is done by placing the camera at the position of the light
- We also modify the Display command to render only the Z buffer
- We also need to change the format to create a square image to capture all the points that the light illuminates
- The information about the camera position must also be stored so we use the MakeShadow command to take this information and write it to the shadow map.

Python Script Design

- To Semi Automate the process A python script is created which will make and render the shadow passes
- All Scene elements are placed in a function which can be passed to the shadow function
- This function moves the scene to be viewed from the light position
- Once the Shadowmaps are created we render the scene again with the shadowspots

```python
def ShadowPass(ri,Name,From,To,coneAngle,SceneFunc) :
  #
  print "Rendering␣Shadow␣pass␣%s.z" %(Name)
  ri.Begin("__render")
  ri.Display(Name+".z", "zfile", "z")
  ri.Clipping(1,10)
  # Specify PAL resolution 1:1 pixel Aspect ratio
  ri.Format(512,512,1)
  # now set the projection to perspective
  ri.Projection(ri.PERSPECTIVE,{ri.FOV:coneAngle*(360/math.pi)})
  #now move to light position
  # create a vector for the Spotlight to and from values

  # to do this we subtract each of thelist elements using a lambda function
  # this is the same as doing the code below I will leave it to you as to which
      you
  # find more readable
  #direction =[To[0]-From[0],To[1]-From[1],To[2]-From[2]]

  direction = map(lambda x,y : x-y , To,From)
  AimZ(ri,direction)
  ri.Translate(-From[0],-From[1],-From[2])
  # now draw the Scene
  ri.WorldBegin()
  SceneFunc(ri)
  ri.WorldEnd()
  ri.MakeShadow(Name+".z",Name+".shad")
  ri.End()
  print "␣Done␣MakeShadow␣%s.shad" %(Name)


```



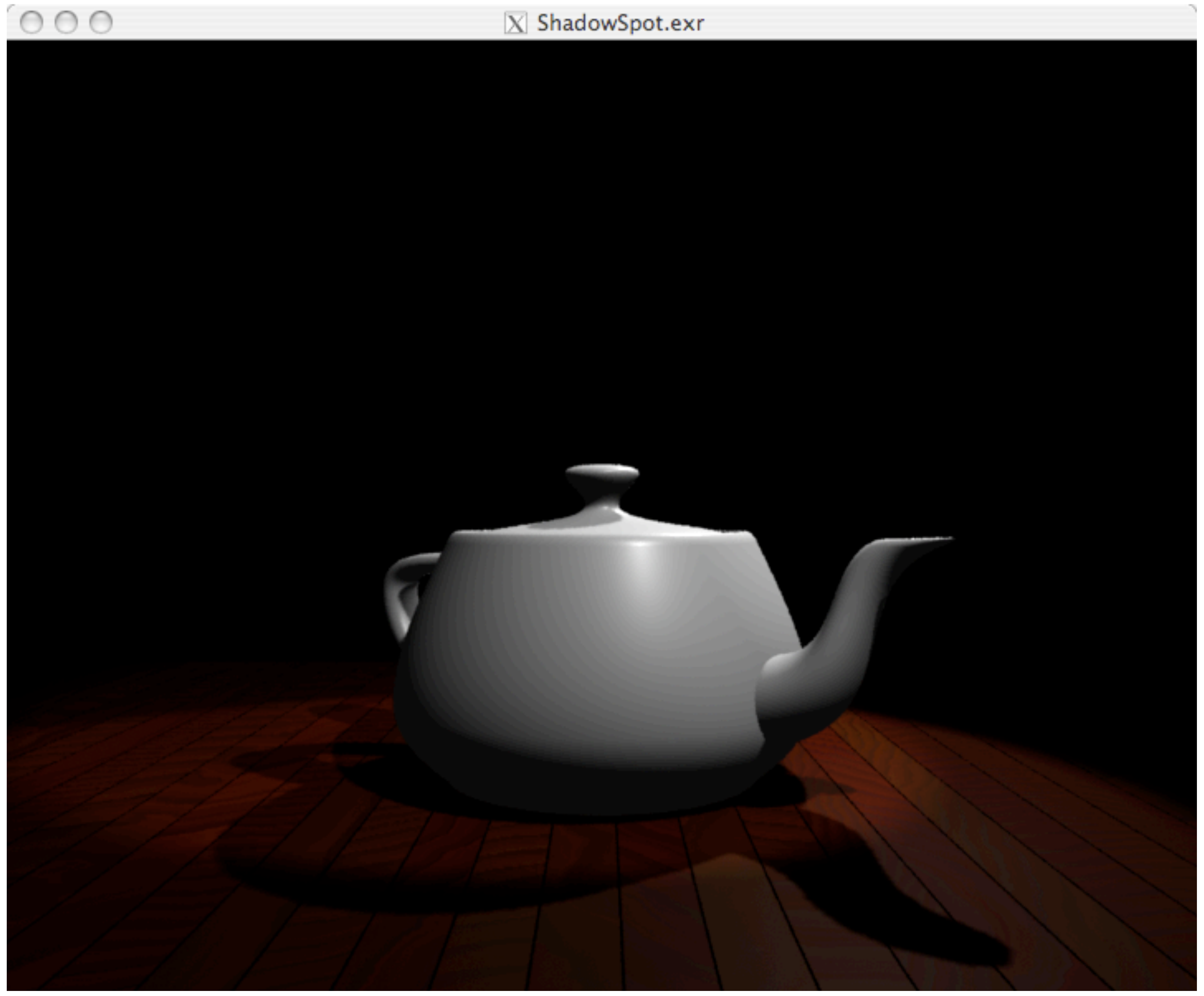
```python
# Modified from Renderman Examples in The renderman Companion
# AimZ(): rotate the world so the direction vector points in
# positive z by rotating about the y axis, then x. The cosine
# of each rotation is given by components of the normalized
# direction vector.  Before the y rotation the direction vector
# might be in negative z, but not afterward.

def AimZ(ri,direction) :
    if (direction[0]==0 and direction[1]==0 and direction[2]==0) :
    return
    # The initial rotation about the y axis is given by the projection of
    # the direction vector onto the x,z plane: the x and z components
    # of the direction.

    xzlen = math.sqrt(direction[0]*direction[0]+direction[2]*direction[2])
    if (xzlen == 0) :
    if(direction[1] <0) :
      yrot = 0
    else :
      yrot =180
    else :
    yrot = 180*math.acos(direction[2]/xzlen)/math.pi;

     # The second rotation, about the x axis, is given by the projection on
     # the y,z plane of the y-rotated direction vector: the original y
     # component, and the rotated x,z vector from above.

    yzlen = math.sqrt(direction[1]*direction[1]+xzlen*xzlen)
    xrot = 180*math.acos(xzlen/yzlen)/math.pi  # yzlen should never be 0

  if (direction[1] > 0) :
    ri.Rotate(xrot, 1.0, 0.0, 0.0)
  else :
    ri.Rotate(-xrot, 1.0, 0.0, 0.0)
  # The last rotation declared gets performed first
  if (direction[0] > 0) :
    ri.Rotate(-yrot, 0.0, 1.0, 0.0)
  else :
    ri.Rotate(yrot, 0.0, 1.0, 0.0)
```

```
Display "Spot1.z" "zfile" "z"
Clipping 1 10
Format 512 512 1
Projection "perspective" "uniform␣float␣fov" [45.8366]
Rotate -47.9689 1 0 0
Rotate 146.31 0 1 0
Translate -2 -4 -3
WorldBegin
  TransformBegin
    AttributeBegin
      Color [1 1 1]
      Translate 0 -1 0
      Rotate -90 1 0 0
      Rotate 36 0 0 1
      Scale 0.4 0.4 0.4
      Surface "plastic"
      Geometry "teapot"
    AttributeEnd

........ [floor geometry]

  TransformEnd

WorldEnd
MakeShadow "Spot1.z" "Spot1.shad"
```

Creating Shadow Passes

```
ri = prman.Ri() # create an instance of the RenderMan interface
ri.Option("rib", {"string asciistyle": "indented"})

SpotFrom=[2,4,3]
SpotTo=[0,0,0]
SpotName="Spot1"
coneAngle=0.4
ShadowPass(ri,SpotName,SpotFrom,SpotTo,coneAngle,Scene)

Spot2From=[2,4,-3]
Spot2To=[0,0,0]
Spot2Name="Spot2"
coneAngle2=0.3
ShadowPass(ri,Spot2Name,Spot2From,Spot2To,coneAngle2,Scene)
```

```
filename = "ShadowSpot.rib"
ri.Begin(filename)
ri.Clipping(1,10)

ri.Display("ShadowSpot.exr", "framebuffer", "rgba")
ri.Format(720,575,1)
ri.Projection(ri.PERSPECTIVE,{ri.FOV:50})

ri.Translate(0,0,4)

ri.WorldBegin()



ri.LightSource ("ambientlight",{ri.HANDLEID: "Ambient","intensity"
    :[0.05]})


ri.LightSource( "shadowspot", {ri.HANDLEID:SpotName,
        "point␣from" : SpotFrom,
              "point␣to" : SpotTo,
              "float␣intensity" : [30],
              "string␣shadowname" :SpotName+".shad",
              "float␣coneangle" : coneAngle,
              "float␣conedeltaangle" : [0.05]})


ri.LightSource( "shadowspot", {ri.HANDLEID:Spot2Name,
        "point␣from" : Spot2From,
              "point␣to" : Spot2To,
              "float␣intensity" : [30],
              "string␣shadowname" :Spot2Name+".shad",
              "float␣coneangle" : coneAngle2,
              "float␣conedeltaangle" : [0.05]})

Scene(ri)

ri.WorldEnd()

ri.End()
```



Using the Shadowmap

- Having created the shadowmap we now return to the original image and use the shadowspot light
- This is exactly the same as the original light only it now uses the shadow map in the "shadowname" parameter

```
LightSource "shadowspot" "Spot1" "float␣coneangle" [0.4]
        "point␣from" [2 4 3] "float␣conedeltaangle" [0.05]
        "string␣shadowname" ["Spot1.shad"]
        "point␣to" [0 0 0] "float␣intensity" [30]

LightSource "shadowspot" "Spot2" "float␣coneangle" [0.3]
        "point␣from" [2 4 -3] "float␣conedeltaangle" [0.05]
        "string␣shadowname" ["Spot2.shad"]
        "point␣to" [0 0 0] "float␣intensity" [30]
```

Ray traced Shadows

- All default lights and shaders can be ray-traced
- However this will increase the render time
- It is still best to use the Shadow map process
- To Use ray tracing we need to enable the following

Visibility Attributes

```
ri.Attribute("visibility", {"int␣diffuse" :1,
              "int␣specular": 1,
              "int␣transmission": 1})

Attribute "visibility" "int␣diffuse" [1] "int␣specular" [1] "int␣transmission" [1]
```

- Diffuse controls the visibility of the current primitive to diffuse rays. These are rays cast by gather, occlusion, and indirectdiffuse.
- Specular controls the visibility of the current primitive to specular rays. These are rays cast by gather, trace, and environment.
- Transmission controls the visibility of primitives to transmission (shadow) rays.

RayTrace Attributes

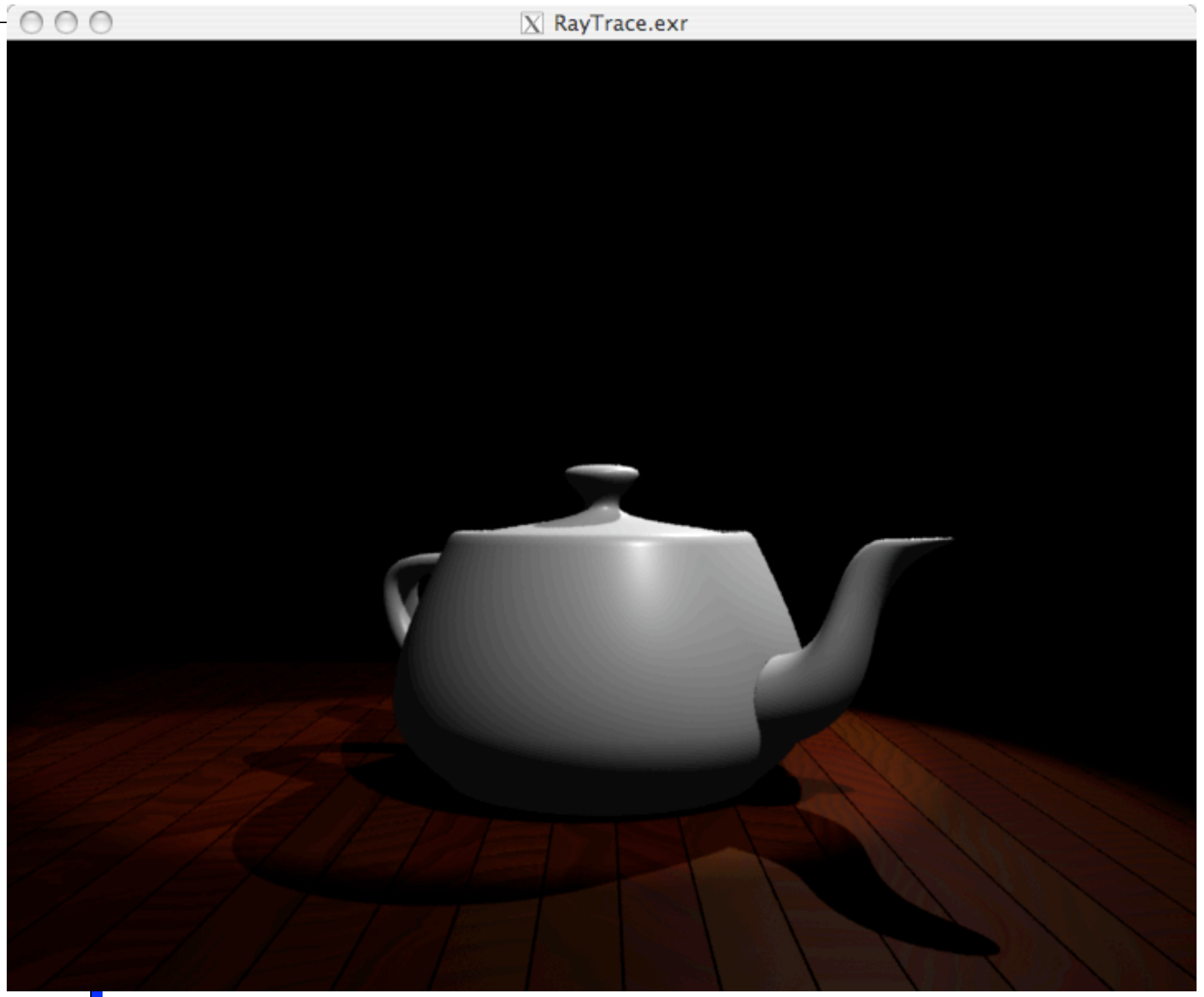
```
ri.Attribute("trace",  {"int␣maxdiffusedepth" :[1], "int␣maxspeculardepth" : [2],
              "int␣displacements" : [0] , "bias" : [.01],
              "int␣samplemotion" : [0]})

 Attribute "trace" "int␣samplemotion" [0] "int␣displacements" [0] "bias" [0.01]
      "int␣maxspeculardepth" [2] "int␣maxdiffusedepth" [1]
```

- these attributes limit the number of bounces (diffuse or specular) for indirect illuminance relative to the associated primitive. To resolve the interaction between per-primitive values when different object have different values of these attributes, we pass the current max down the ray tree and calculate maxdiffusedepth = MIN(parent.maxdiffusedepth, parent.diffusedepth + object.maxdiffusedepth) -- and similar for maxspeculardepth. For photon tracing, the maxdiffusedepth option is used to limit the number of diffuse bounces of photons, while maxspeculardepth limits the number of specular reflection and refraction bounces.

Raytrace Attributes

- displacements controls whether true displacements appear in ray traced results. When false, the displacement will be disregarded for the purposes of ray-primitive intersection tests, but shading will take the displacements into account effectively resulting in a bump mapped appearance.
- the bias value affects transmission/shadow rays as well as trace/environment rays. It is an offset applied to the ray origin, moving it slightly away from the surface launch point in the ray direction. This offset can prevent blotchy artefacts resulting from the ray immediately finding an intersection with the surface it just left.
- samplemotion controls whether motion blurred objects appear in ray traced results. When 0, the motion blur of other objects hit by rays launched from the object with this attribute will be ignored. When non-zero, motion blur will be taken into account by rays launched from an object with this attribute.

```

ri = prman.Ri() # create an instance of the RenderMan interface

filename = "RayTrace.rib"

ri.Begin(filename)
ri.Clipping(1,10)

ri.Attribute("visibility", {"int diffuse" :1,
              "int specular": 1,
              "int transmission": 1})

ri.Attribute("trace",  {"int maxdiffusedepth" :[1], "int maxspeculardepth" : [2],
              "int displacements" : [0] , "bias" : [.01],
              "int samplemotion" : [0]})

# now we add the display element using the usual elements
# FILENAME DISPLAY Type Output format
ri.Display("RayTrace.exr", "framebuffer", "rgba")
# Specify PAL resolution 1:1 pixel Aspect ratio
ri.Format(720,575,1)
# now set the projection to perspective
ri.Projection(ri.PERSPECTIVE,{ri.FOV:50})

ri.Translate(0,0,4)
# now we start our world
ri.WorldBegin()

ri.LightSource ("ambientlight",{ri.HANDLEID: "Ambient","intensity" :[0.05]})
ri.LightSource( "shadowspot", {ri.HANDLEID:SpotName,
        "point from" : SpotFrom,
              "point to" : SpotTo,
              "float intensity" : [30],
              "string shadowname" :"raytrace",
              "float coneangle" : coneAngle,
              "float conedeltaangle" : [0.05]})

ri.LightSource( "shadowspot", {ri.HANDLEID:Spot2Name,
        "point from" : Spot2From,
              "point to" : Spot2To,
              "float intensity" : [30],
              "string shadowname" :"raytrace",
              "float coneangle" : coneAngle2,
              "float conedeltaangle" : [0.05]})
Scene(ri)

ri.WorldEnd()

# and finally end the rib file
ri.End()
```



```
##RenderMan RIB
version 3.04
Clipping 1 10
Attribute "visibility" "int␣diffuse" [1] "int␣specular" [1] "int␣transmission" [1]
Attribute "trace" "int␣samplemotion" [0] "int␣displacements" [0] "bias" [0.01]
      "int␣maxspeculardepth" [2] "int␣maxdiffusedepth" [1]
#File RayTrace.rib
#Created by jmacey
#Creation Date: Tue Sep 30 15:17:14 2008
Declare "Spot1" "string"
Declare "Ambient" "string"
Display "RayTrace.exr" "framebuffer" "rgba"
Format 720 575 1
Projection "perspective" "uniform␣float␣fov" [50]
Translate 0 0 4
WorldBegin
  LightSource "ambientlight" "Ambient" "intensity" [0.05]
  LightSource "shadowspot" "Spot1" "float␣coneangle" [0.4] "point␣from" [2 4 3]
        "float␣conedeltaangle" [0.05] "string␣shadowname" ["raytrace"]
        "point␣to" [0 0 0] "float␣intensity" [30]
  LightSource "shadowspot" "Spot2" "float␣coneangle" [0.3] "point␣from" [2 4 -3]
        "float␣conedeltaangle" [0.05] "string␣shadowname" ["raytrace"]
        "point␣to" [0 0 0] "float␣intensity" [30]
  TransformBegin
    AttributeBegin
      Color [1 1 1]
      Translate 0 -1 0
      Rotate -90 1 0 0
      Rotate 36 0 0 1
      Scale 0.4 0.4 0.4
      Surface "plastic"
      Geometry "teapot"
      ................
```

Placing the Camera

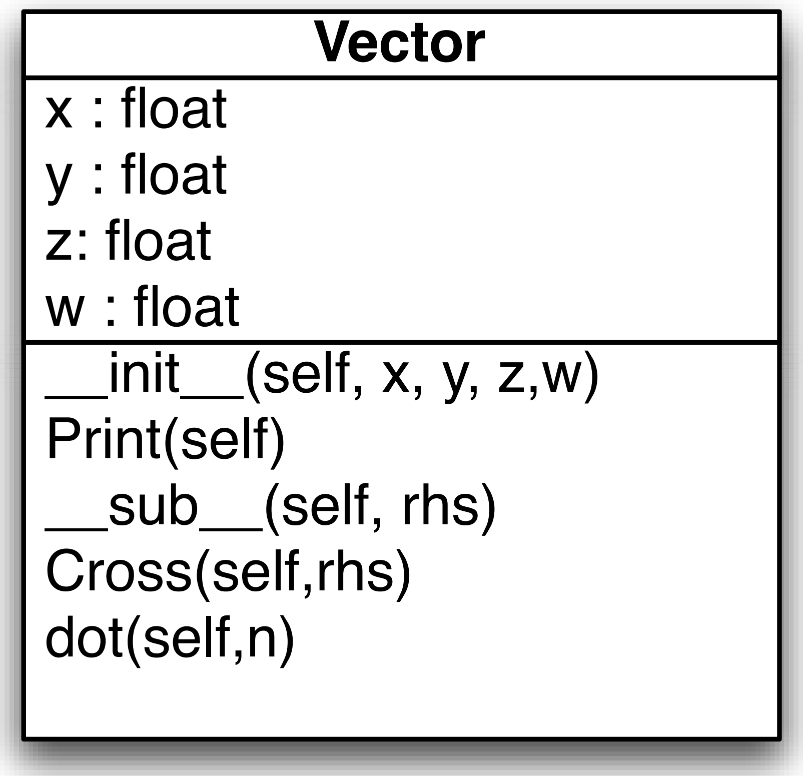
- It is sometimes difficult to visualise the positioning of the Camera at the light source.
- To make this easier a python class has been developed to encapsulate the camera functionality
- This will generate a transformation matrix which can be used to position the camera in the scene.
- As well as set other camera features

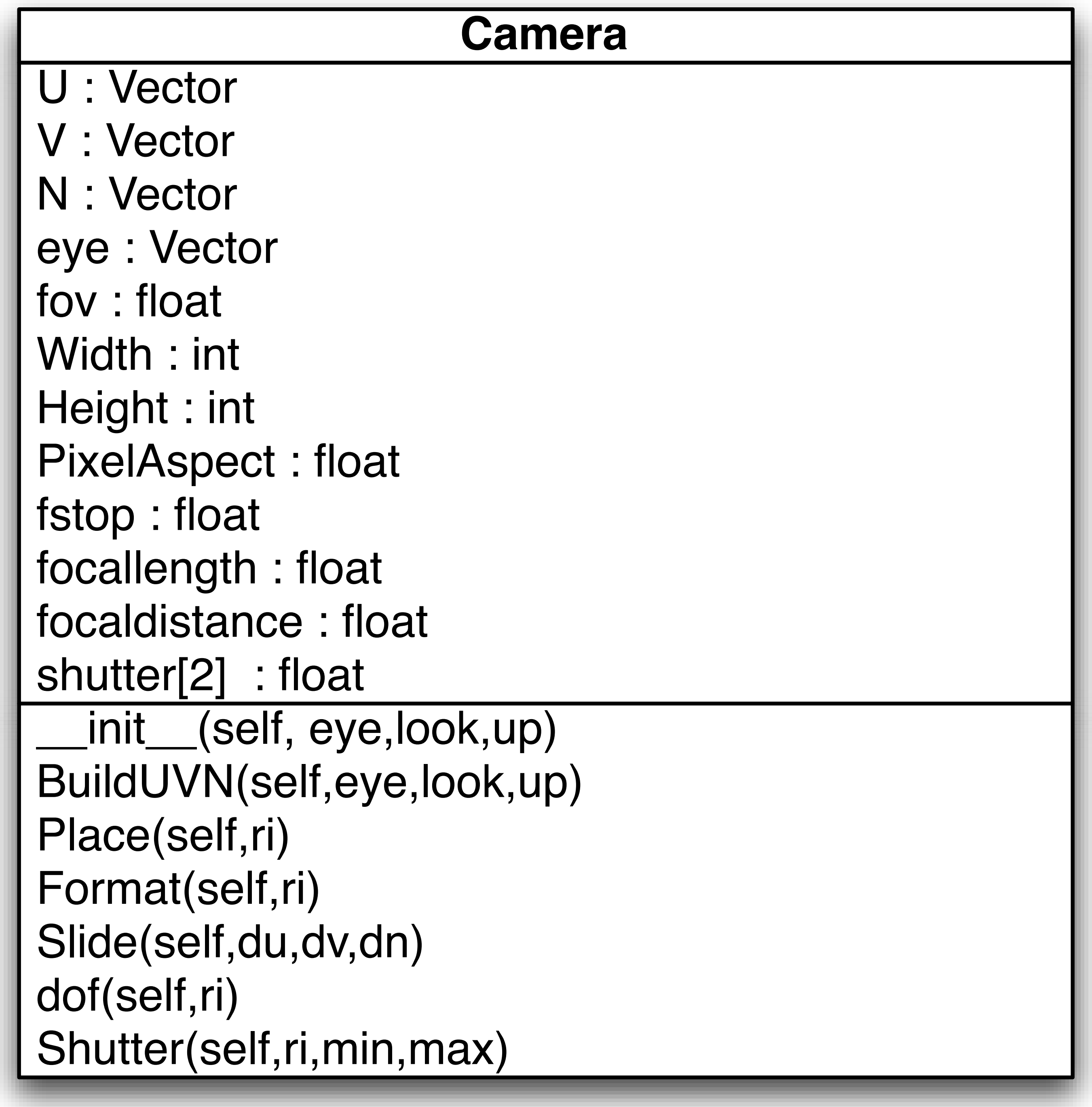
Python Path

- To make these classes available to other python modules I have placed them in a directory called PythonClasses
- We need to tell the python interpreter where to find these classes which can be done in two ways

```
# in .bashrc add the path to the files

export PYTHONPATH=$PYTHONPATH:~/Renderman/Example/PythonClasses

# or in a python script add

import sys
sys.path.append("~/Renderman/Example/PythonClasses")
```

Camera Class



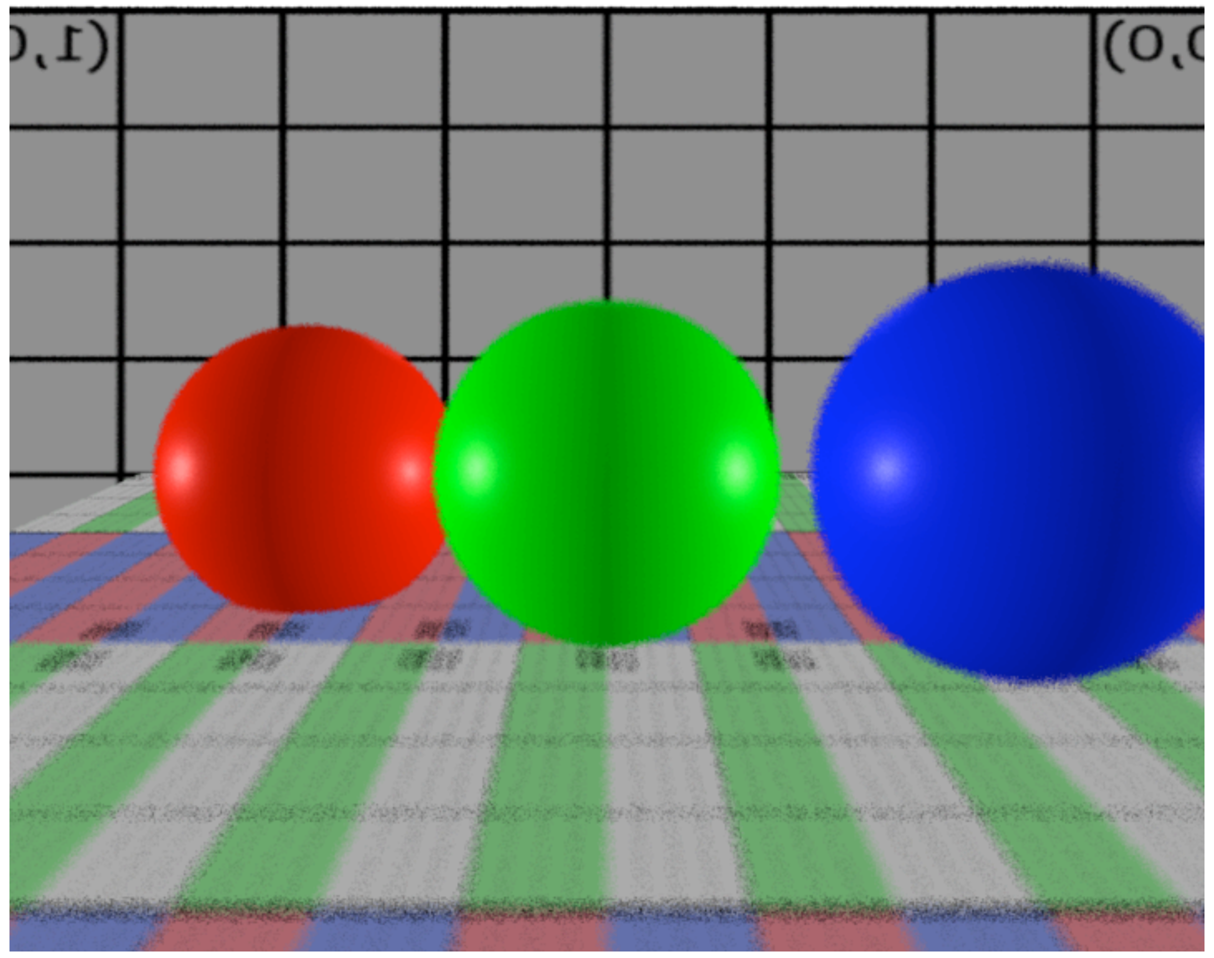


```python
class Camera :
  U=Vector(1,0,0,1)
  V=Vector(0,1,0,1)
  N=Vector(0,0,1,1)
  eye=Vector(0,0,0,1)
  fov=50
  Width=720
  Height=576
  PixelAspect=1.0
  fstop=16
  focallength=8
  focaldistance=10
  shutter =[0,1]
  def __init__(self, eye,look,up):
    # now construct the cameras viewing vectors N is eye-look
    self.eye=eye
    self.BuildUVN(eye,look,up)


  def BuildUVN(self,eye,look,up) :
    self.N=eye-look
    # now construct another orthogonal to the N
    self.U=up.Cross(self.N)
    # and finally the  new up vector
    self.V=self.N.Cross(self.U)
    # normalize the vectors to unit length
    self.N.normalize()
    self.U.normalize()
    self.V.normalize()
```

```python
  def Place(self,ri):
    U=self.U
    V=self.V
    N=self.N
    eye=self.eye
    ri.Identity();
    ri.Scale(1,1,-1)
    tx=[U.x,V.x,N.x,0.0,U.y,V.y,N.y,0.0,U.z,V.z,N.z,0.0,-eye.dot(U
        ),-eye.dot(V),-eye.dot(N),1.0]
    ri.ConcatTransform(tx)

  def Format(self,ri) :
    ri.Format(self.Width,self.Height,self.PixelAspect)
    # now set the projection to perspective
    ri.Projection(ri.PERSPECTIVE,{ri.FOV: self.fov})

  def Slide(self,du,dv,dn) :
    self.eye.x += du * self.U.x + dv * self.V.x + dn * self.N.x;
    self.eye.y += du * self.U.y + dv * self.V.y + dn * self.N.y;
    self.eye.z += du * self.U.z + dv * self.V.z + dn * self.N.z;

  def dof(self,ri) :
    ri.DepthOfField(self.fstop,self.focallength,self.focaldistance
        )

  def Shutter(self,ri,min,max) :
    self.shutter=[min,max]
    ri.Shutter(min,max)
```

```python
ri = prman.Ri() # create an instance of the RenderMan interface

# build the camera
cam=Camera(Vector(4,0.2,4,1),Vector(0,0,0,1),Vector(0,1,0,0))
cam.fov=40

filename = "Camera.rib"

ri.Begin(filename)


# now we add the display element using the usual elements
# FILENAME DISPLAY Type Output format
ri.Display("Camera.exr", "framebuffer", "rgba")

# create the screen format
cam.Format(ri)

# now we start our world
ri.WorldBegin()
# now place our camera in the scene
cam.Place(ri)
ri.TransformBegin()
Scene(ri)
ri.TransformEnd()
# end our world
ri.WorldEnd()
# and finally end the rib file
ri.End()
```

```
Display "Camera.exr" "framebuffer" "rgba"
Format 720 576 1
Projection "perspective" "uniform␣float␣fov" [40]
WorldBegin

Identity
Scale 1 1 -1
ConcatTransform [ 0.707107 -0.0249844 0.706665 0  0 0.999376 0.0353333 0  -0.707107
         -0.0249844 0.706665 0  -0 2.77556e-17 -5.66039 1 ]
TransformBegin
```

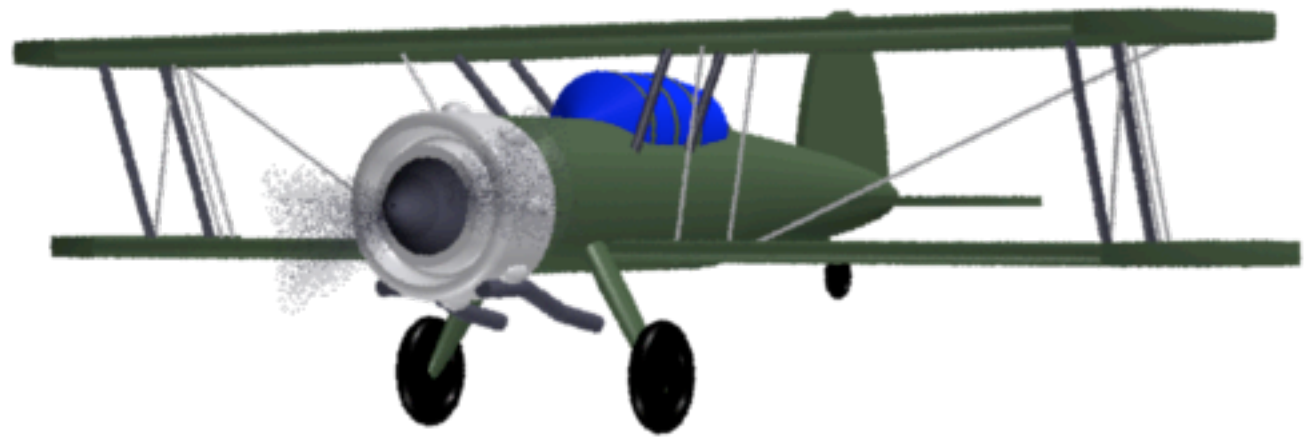


Depth of Field

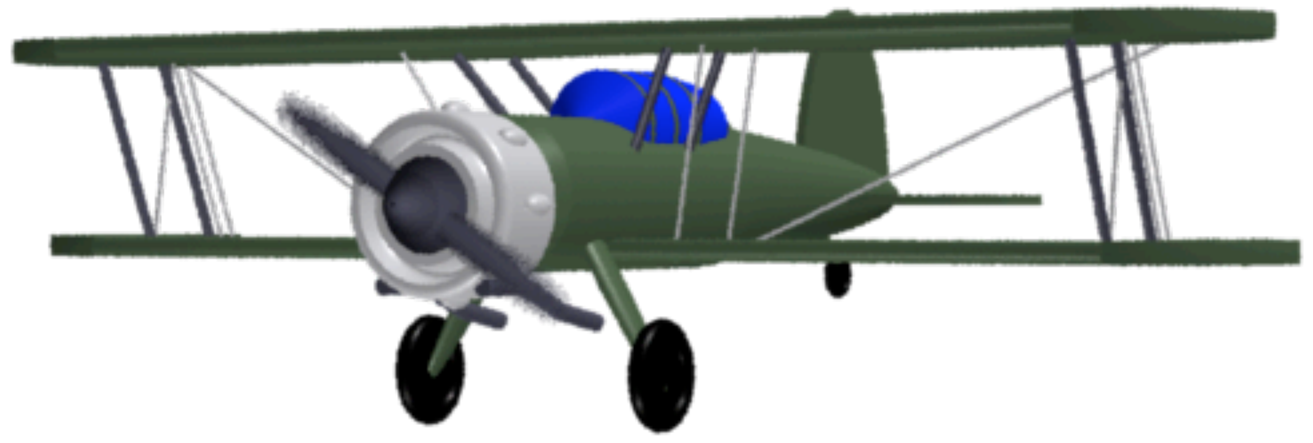
```
ri.DepthOfField(fstop,focallength,focaldistance)

DepthOfField 22 45 1200
```

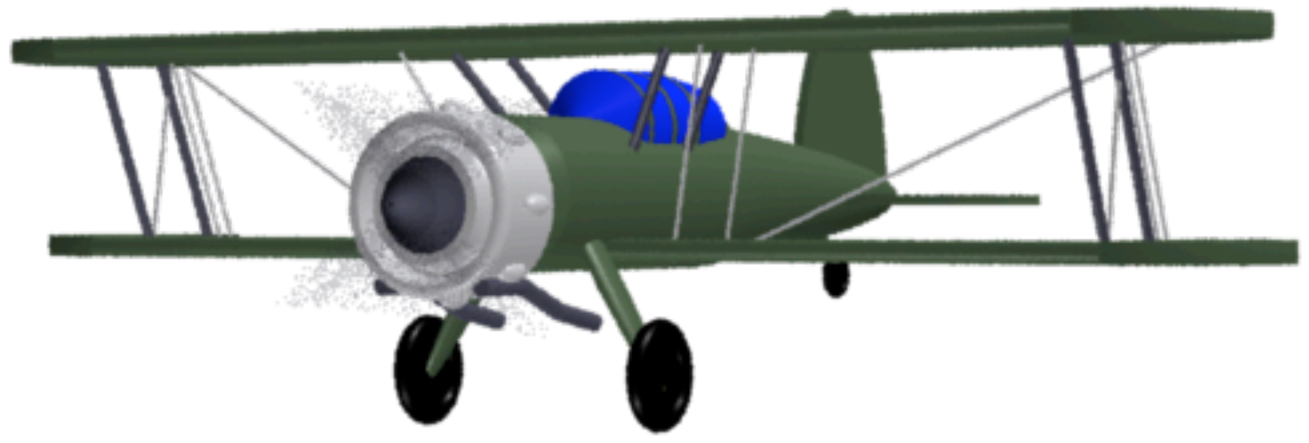
- Renderman allows modelling of depth of field by the use of the DepthOfField Function
- We can specify the aperture of the lens by setting the fstop
- Then the focal length and focal distance.
- Element outside of the focal area will be rendered with blur to simulate camera lens out of focus areas

Motion Blur



- Some rendering programs are capable of performing temporal anti-aliasing and motion blur.
- Motion blur is specified through moving transformations and moving geometric primitives.
- Appearance parameters, such as colour, opacity, and shader variables can also be changed during a frame.
- To specify objects that vary over time, several copies of the same object are created, each with different parameters at different times within a frame.
- The times that actually contribute to the motion blur are set with the RiShutter command.
- Parameter values change linearly over the intervals between knots. There is no limit to the number of time values associated with a motion-blurred primitive, although two is usually sufficient.





Motion Blur

- Rigid body motions and other transformation-based movements are modeled using moving coordinate systems.
- Moving coordinate systems are created by giving a sequence of transformations at different times and can be concatenated and nested hierarchically.
- All output primitives are defined in the current object coordinate system and, if that coordinate system is moving, the primitives will also be moving.
- Moving geometry is created by bracketing the definitions at different times between RiMotionBegin and RiMotionEnd calls.

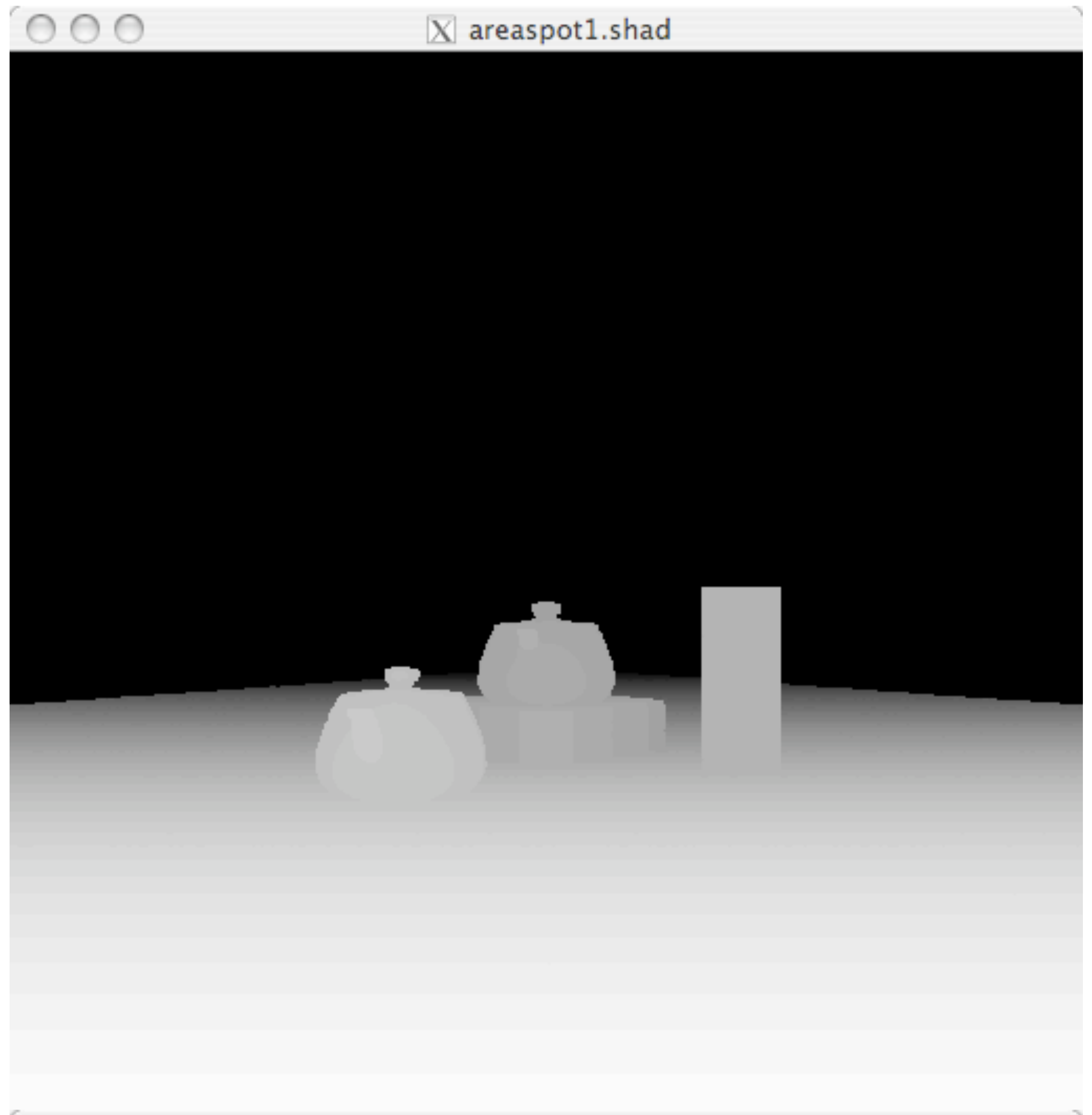
```python
def Prop(rotation,Next) :
  ri.Rotate(90,0,0,1)
  # read in the nose cone data no need to blur
  ri.ReadArchive("Prop.rib")
  # this is our main interpolated rotation
  ri.MotionBegin([0,1])
  ri.Rotate(rotation,0,0,1)
  ri.Rotate(rotation+Next,0,0,1)
  ri.MotionEnd()
  # now we draw all the prop sections one at a time
  # note the transformbegin / end sections are
      outside the motion blocks
  # prop 1
  ri.TransformBegin()
  ri.MotionBegin([0,1])
  ri.Translate( 0 ,0.8, 0.4)
  ri.Translate( 0 ,0.8, 0.4)
  ri.MotionEnd()

  ri.MotionBegin([0,1])
  ri.Scale( 1 ,6 ,0.5)
  ri.Scale( 1 ,6 ,0.5)
  ri.MotionEnd()

  ri.MotionBegin([0,1])
  ri.Rotate( 10 ,0 ,1 ,0)
  ri.Rotate( 10 ,0 ,1 ,0)
  ri.MotionEnd()

  ri.MotionBegin([0,1])
  ri.Sphere( 0.15, -0.15, 0.15, 360)
  ri.Sphere( 0.15, -0.15, 0.15, 360)
  ri.MotionEnd()

  ri.TransformEnd()

```

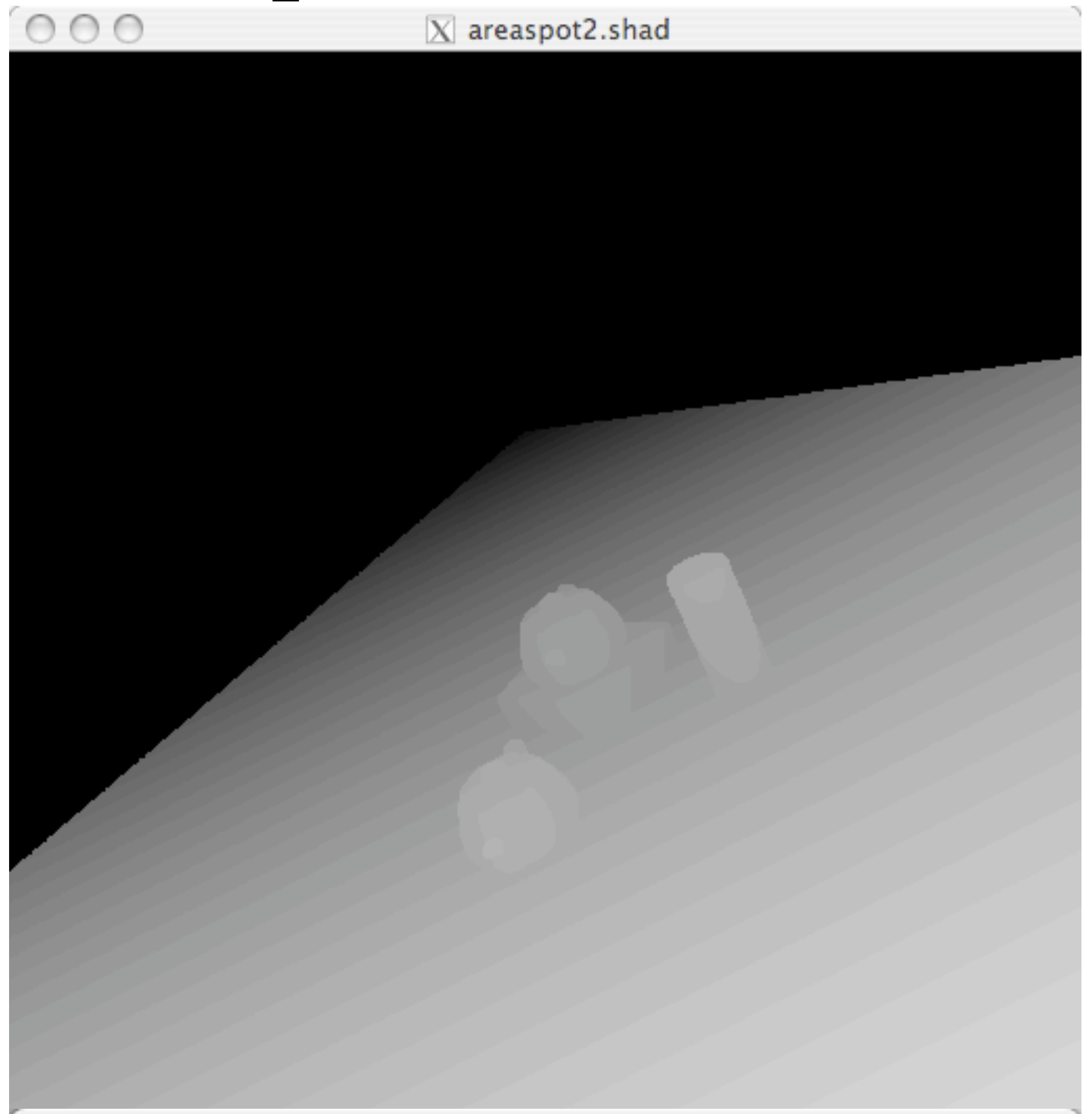
```
MotionBegin [0 1]
  Rotate 40 0 0 1
  Rotate 85 0 0 1
MotionEnd
TransformBegin
  MotionBegin [0 1]
    Translate 0 0.8 0.4
    Translate 0 0.8 0.4
  MotionEnd
  MotionBegin [0 1]
    Scale 1 6 0.5
    Scale 1 6 0.5
  MotionEnd
  MotionBegin [0 1]
    Rotate 10 0 1 0
    Rotate 10 0 1 0
  MotionEnd
  MotionBegin [0 1]
    Sphere 0.15 -0.15 0.15 360
    Sphere 0.15 -0.15 0.15 360
  MotionEnd
TransformEnd
```

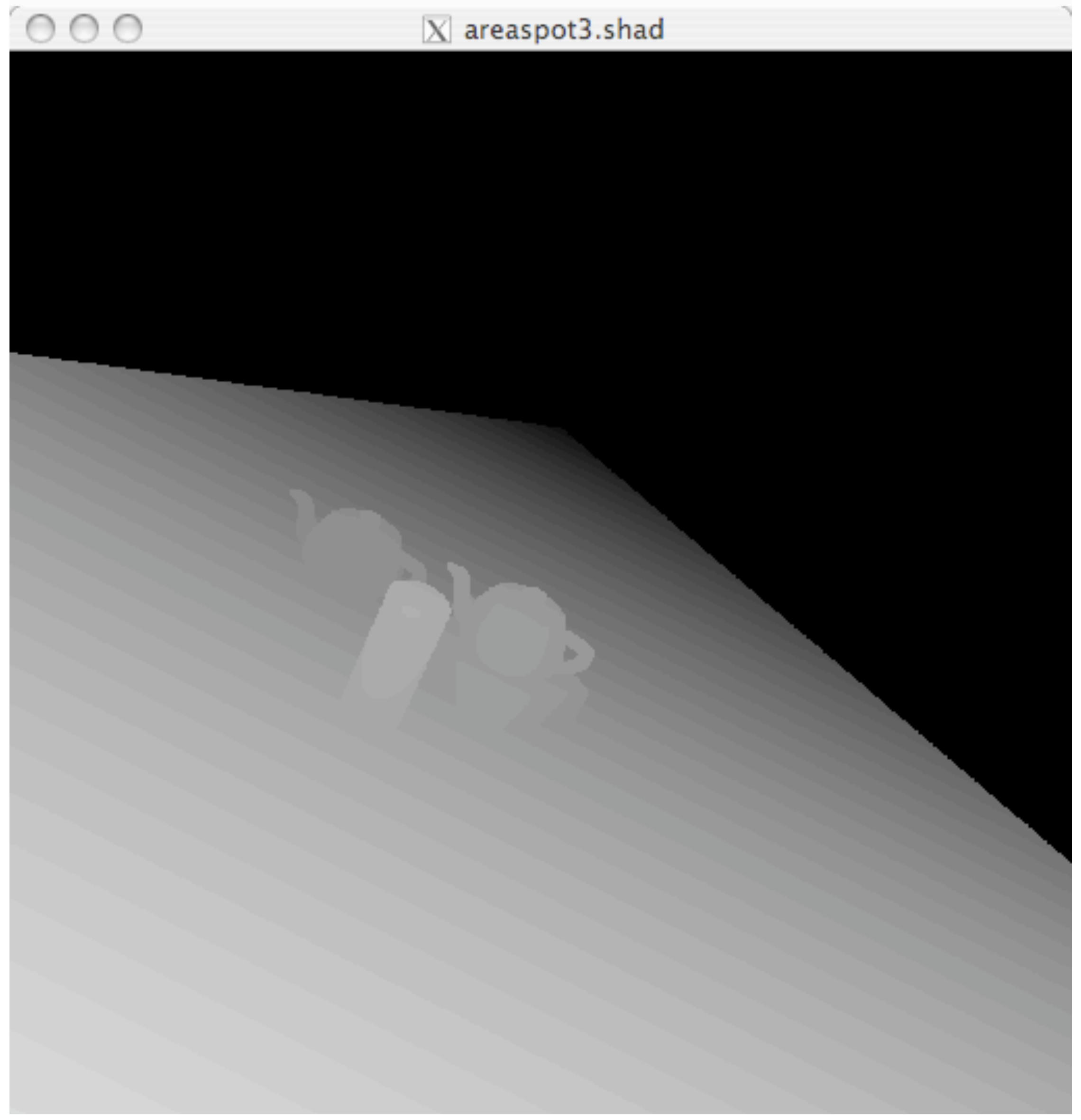
Area Lights

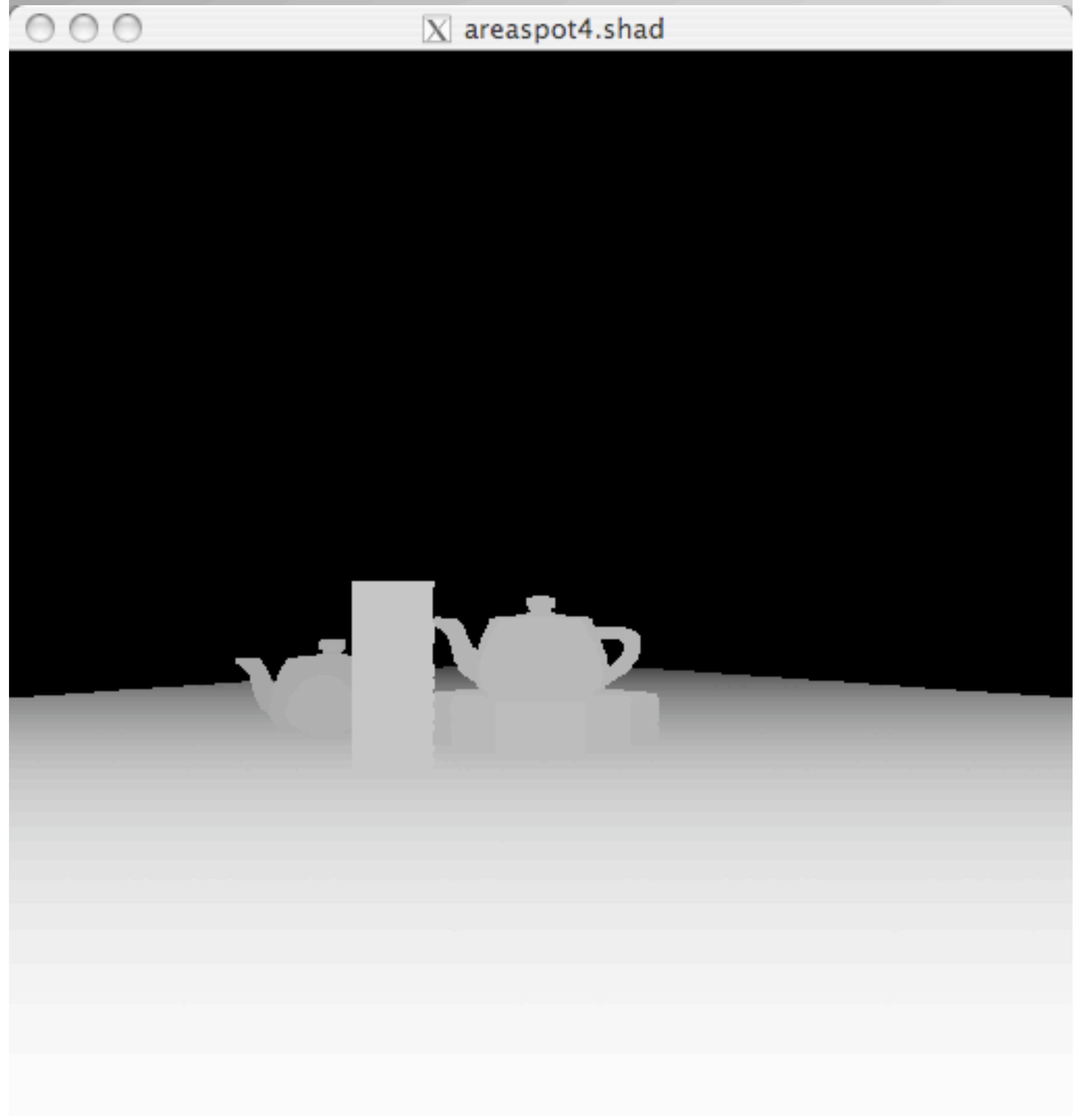
- Renderman has no direct support for area lights
- However it is easy to implement one using a number of shadow maps and averaging the contribution of each of the shadows
- We can do this by writing a simple light shader
- This will allow us to create soft shadows

Area Lights Step 1





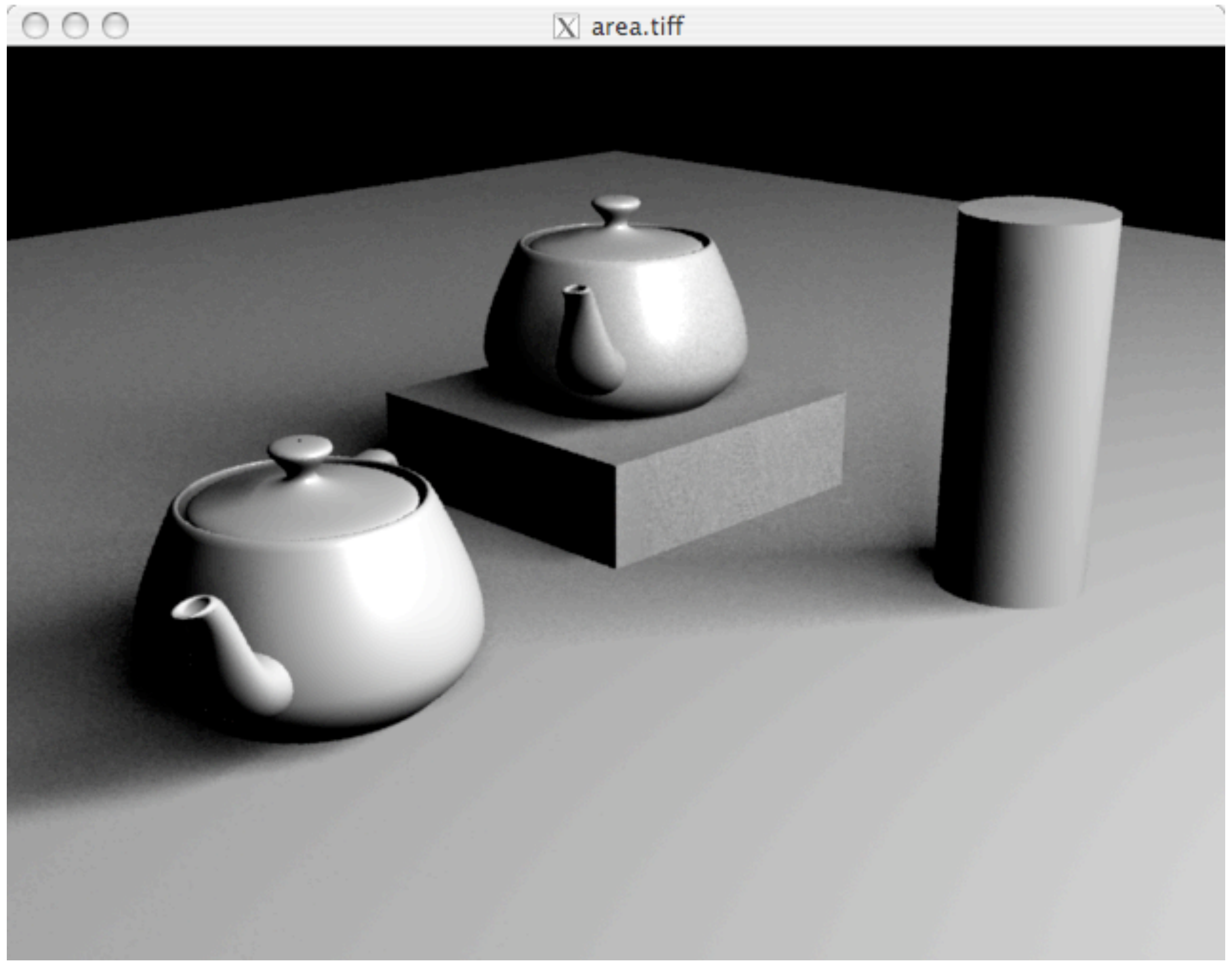




Area Light Shader

```
light arealight(color lightcolor = color(1,1,1);
    float intensity = 1;
    string maplist = "";
    float numsamples = 1;
    point Pl1 = point(0, 0, 0);
    point Pl2 = point(0, 1, 0);
    point Pl3 = point(0, 0, 1);
    point Pl4 = point(0, 1, 1);
    float shadowBias = 0.001;
    float gapBias = 0.01) {
    varying float attenuation;

    illuminate ((Pl1+Pl2+Pl3+Pl4)*0.25) {   // base illumination at average
                                             // of light positions

    attenuation = shadow(maplist,Ps,"source",Pl1,Pl2,Pl3,Pl4,
              "samples",numsamples, "bias",shadowBias, "gapbias", gapBias);

    Cl = lightcolor * intensity * (1-attenuation);
  }
}
```

AreaLight



```
LightSource "arealight" "areaLight" "intensity" [2]
      "maplist" ["areaspot1.shad,areaspot2.shad,areaspot3.shad,areaspot4.shad"]
      "Pl1" [5 0 5]
      "Pl2" [5 5 5]
      "Pl3" [5 0 -5]
      "Pl4" [5 5  -5]
      "gapBias" [0.5]
      "shadowBias" [0.2]
      "numsamples" [36]
```

References

- [1] Ian Stephenson. Essential Renderman Fast. Springer-Verlag, 2003.
- [2] Larry Gritz Anthony A Apodaca. Advanced Renderman (Creating CGI for Motion Pictures). Morgan Kaufmann, 2000.
- Renderman Documentation Appendix D - RenderMan Interface Bytestream Conventions
- Application Note #26 Soft Shadows in PhotoRealistic RenderMan
- Renderman Documentation Motion