

Advanced Lighting – casting shadows where the sun never shines!

Overview

Shadows in ‘real life’ are normally considered to be an integral part of scene – a natural outcome of the interplay of light and objects. In 3D graphics, however, a shadow is treated as if it were an additional element. Infact, shadow casting is an extra facility that must be explicitly ‘turned on’.

When lit by one of the **basic light sources**, objects are shaded according to the orientation and optical characteristics of their surfaces. But basic light sources are unable to **cast** shadows of objects onto other objects. With the exception of an ambientlight, each of the basic light sources has a counter-part that allows shadow casting to occur. For example, instead of using a basic distant light in a scene, such as

```
LightSource "distant" 1 "from" [3 3 3] "to" [0 0 0]
```

its shadow casting equivalent could be used ie.

```
LightSource "shadowdistant" 1 "from" [3 3 3] "to" [0 0 0] "shadowname" "shadow1.tx"
```

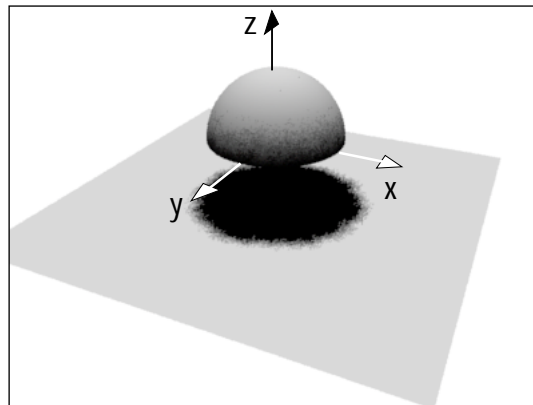
For a complete description of the shadowdistant and other shadow casting light sources refer to Appendix C – Shader Reference, pages 12 to 14.

Shadow casting is not an automatic attribute of every light source because the renderer is required to perform additional calculations in order to determine the location and interaction of the shadows. This makes the rendering process slow and, in some situations, it is not always necessary to have shadows. Unlike the technique known as ray-tracing, in which the shadows are created along with every other part of an image, RenderMan’s so-called scan-line renderer calculates the contribution that each “shadow casting” light source will make to the final image BEFORE it is rendered. This method of creating shadows is more efficient than ray-tracing if, for example, over part of an animation the lights and objects in a scene remain in a fixed relationship to each other. During such periods of ‘static’ lighting the renderer need only perform the lengthy shadow calculations on the first frame, there after it can apply the pre-calculated shadow information to all subsequent frames relatively quickly.

Pre-processing the shadows before making the final image also offers more creative control. For example, the renderer can be forced to use the “wrong” shadow information. In this way light sources can be made to cast shadows around corners, or an object can cast the shadow of an entirely different object. Such manipulations are beyond the capabilities of most ray-tracers.

For each light source that will create a shadow(s) in a scene, RenderMan requires a special “image” to be generated from the view-point of the light source. However, these special output files do not contain images as such, but store information about how far away each part of the scene is from the light source.

The following example is a very simple scene lit by a single (shadowing) distant light.



```
LightSource shadowdistant "intensity" 1.5 "from" [0 0 4]
"to" [0 0 0] "shadowname" "shadow.tx"
```

step 1 The first step in creating shadows is the production of a depth map for each (shadow) light source. Because the output file from this step only contains depth information it is unnecessary, when defining the scene from the view point of a light, to specify the colour or surface properties of the objects themselves, for example,

```
FrameBegin 1
  Display "depth.pic" "zfile" "z"
  Format 128 128 1
  Projection "perspective" "fov" 110

  Translate 0 0 4 # equivalent to moving the view-point 4
  Rotate 180 0 1 0 # units directly above the origin of the scene

  WorldBegin
    ObjectInstance 1
    ObjectInstance 2
  WorldEnd
  MakeShadow "depth.pic" "shadow.tx"
FrameEnd
```

Notice how this first step is contained within its own frame block. This isolates it from remainder of the RIB script that produces the final full coloured rendered image.

step 1 – continued The first frame produces two files, namely, “depth.pic” and “shadow.tx”. The display statement that produces “depth.pic” uses two new parameters,

```
Display "depth.pic" "zfile" "z"
```

The inclusion of the letter “z” indicates to the renderer that it is to produce a depth map rather than a normal full colour image. Step 1 is concluded with the production of a texture file from the depth map,

```
MakeShadow "depth.pic" "shadow.tx"
```

step 2 In the second step the texture file(s) produced in step 1 is (are) used by the appropriate light source(s) to calculate the correct lighting values for each part of the scene, for example,

```
FrameBegin 2
  Display "half ball.tiff" "tiff" "rgba"
  Format 400 300 1
  Projection "perspective" "fov" 40

  Translate 0 1 8
  Rotate -120 1 0 0
  Rotate 25 0 0 1

  WorldBegin
    LightSource "shadowdistant" 1 "intensity" 1.5
    "from" [0 0 4] "to" [0 0 0] "shadowname" "shadow.tx"
    Color .5 .5 .5
    Surface "matte"
    ObjectInstance 1
    Color .5 .5 .5
    ObjectInstance 2
  WorldEnd
FrameEnd
```

There are several points to be noted in this example. Strictly speaking the RIB script, for the sake of simplicity, has produced an incorrect shadow! Distant light sources behave much like the sun – they produce shadows with parallel light. In the first frame the depth map was made using perspective projection with a field of view large enough to “see” the entire scene.

```
Projection "perspective" "fov" 110
```

Without this simplification it would have been necessary to use orthographic projection and to scale the scene in order for it to “fit” into a viewing space 1 unit by 1 unit – the dimensions of an orthographic viewing frame.

Alternatively, if the shadow version of a pointlight had been used instead of a shadowdistant it would have been necessary to produce 6 depth maps – each one corresponding to the 6 directions a point light source can radiate light ie. top, bottom, left, right, front and back! The important point is that although the final image is technically incorrect it is still CONVINCING.

Example 1
complete script

RIB script

```
# Experiments with single shadows

ObjectBegin 1
  Sphere 1 0 1 360
ObjectEnd
ObjectBegin 2
  Polygon "P" [-3 3 -1 -3 -3 -1 3 -3 -1 3 3 -1 ]
ObjectEnd

FrameBegin 1
  Display "depth.pic" "zfile" "z"
  Format 128 128 1
  Projection "perspective" "fov" 110

  Translate 0 0 4
  Rotate 180 0 1 0

  WorldBegin
    ObjectInstance 1
    ObjectInstance 2
  WorldEnd
  MakeShadow "depth.pic" "shadow.tx"
FrameEnd

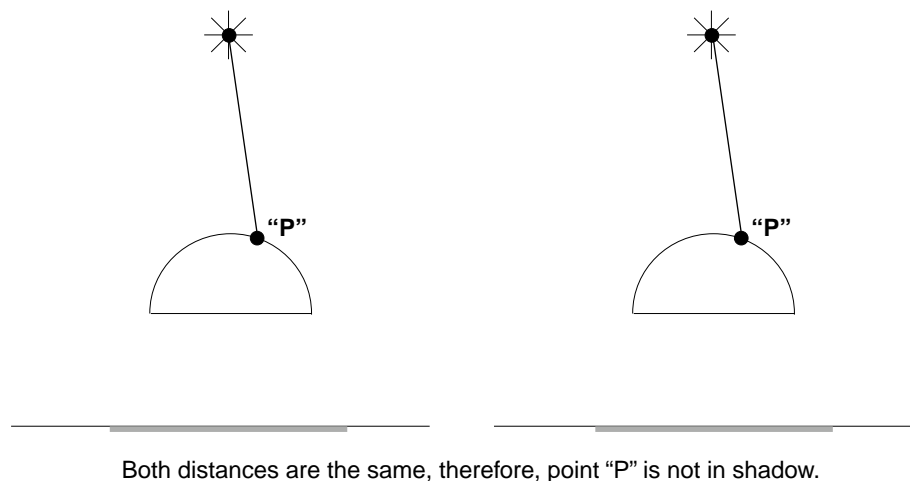
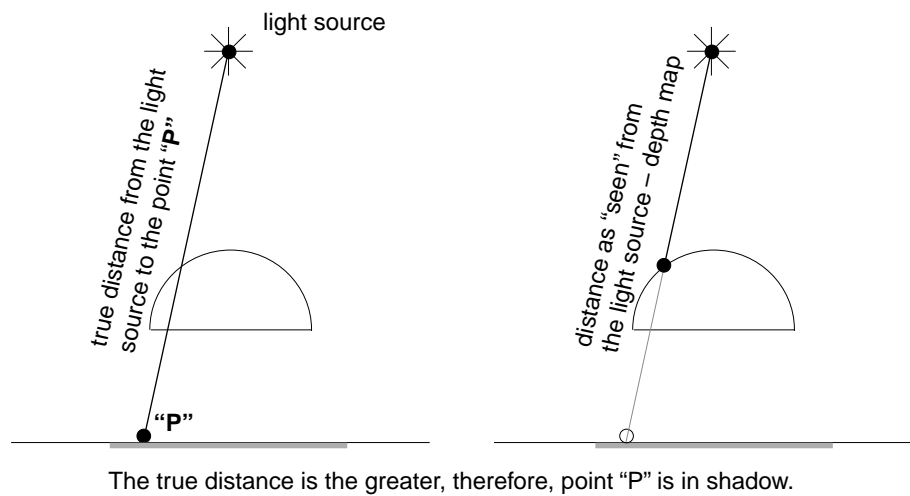
FrameBegin 2
  Display "half ball.tiff" "tiff" "rgba"
  Format 400 300 1
  Projection "perspective" "fov" 40

  Translate 0 1 8
  Rotate -120 1 0 0
  Rotate 25 0 0 1

  WorldBegin
    LightSource "shadowdistant" 1 "intensity" 1.5 "from" [0 0 4]
    "to" [0 0 0] "shadowname" "shadow.tx"
    Color .5 .5 .5
    Surface "matte"
    ObjectInstance 1
    Color .5 .5 .5
    ObjectInstance 2
  WorldEnd
FrameEnd
```

The Shadow Algorithm
– how it works

When using a light source that creates shadows the renderer determines if a point in a 3D scene lies within a shadow cast by another object by comparing two distances. Firstly, it calculates the true distance from the point to the light source and then it compares this value to the corresponding distance in the texture file that was produced from a depth map. If the true distance is the larger of the two then the screen pixel corresponding to the 3D point is shaded a dark colour appropriate to a shadow. Alternatively, if the true distance is smaller then the screen pixel is assigned the colour and brightness of the corresponding 3D point on the surface of the object casting the shadow.



Using FrameUP to animate a scene with shadows

An example animation

The next two pages list a sample file that can be read and converted to an animation RIB file by the utility program FrameUP. Unlike the animation files found in the previous section each KeyFrameBegin/KeyFrameEnd block contains two frames. The first creates the texture file used by the shadowdistant light found in the second frame of each of the two key frames.

RIB script

```
#Example file for creating shadows using FrameUP
Option "limits" "bucketsize" [32 32]
ShadingRate 4
Display "test1""file" "rgb"

ObjectBegin 1
  Polygon "P" [-3 3 -1 -3 -3 -1 3 -3 -1 3 3 -1]
ObjectEnd
ObjectBegin 2
  Cone 2 1 360
ObjectEnd

Tween "from" 1 "to" 2 "frames" 100 "smooth"

KeyFrameBegin 1
  FrameBegin
    Display "shadow1.pic" "zfile" "z"
    Format 512 512 1
    Projection "perspective" "fov" 90

    #Position of shadowlight 1
    Rotate -153.4 1 0 0
    Rotate -90.0 0 0 1
    Translate 2.0 0.0 -4.0

    WorldBegin
      ObjectInstance 1
      Rotate 0 1 1 0
      ObjectInstance 2
    WorldEnd
    MakeShadow "shadow1.pic" "shadowmap1.tx"
  FrameEnd
  FrameBegin
    Display "test1""file" "rgb"
    Format 400 320 1
    Projection "perspective" "fov" 40
    Translate 0 0 12
    Rotate -120 1 0 0
    Rotate 25 0 0 1
    WorldBegin
      LightSource "shadowdistant" 1 "intensity" 2
      "from" [-2 0 4] "to" [0 0 0] "shadowname" "shadowmap1.tx"
      LightSource "ambientlight" 2 "intensity" 0.2
      Surface "plastic"
      Color 1 0 0
      ObjectInstance 1
      Color 1 1 0
      Rotate 0 1 1 0
      ObjectInstance 2
    WorldEnd
  FrameEnd
KeyFrameEnd
```

calculate the shadow information for the shadowdistant light in key frame number 1

use the shadow information...

An example Animation
– continued

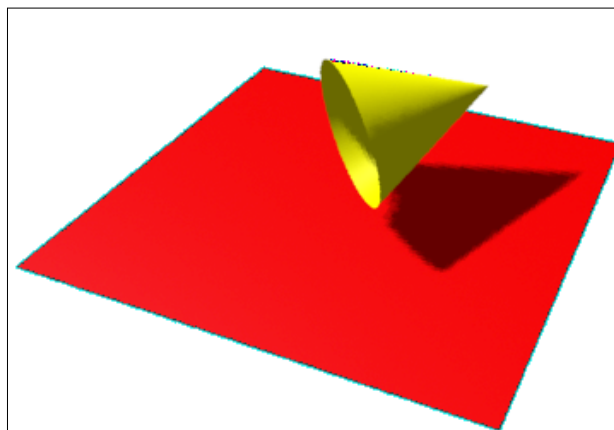
calculate the shadow information for the shadowdistant light in key frame number 2

use the shadow information...

```
KeyFrameBegin 2
FrameBegin
  Display "shadow1.pic" "zfile" "z"
  Format 512 512 1
  Projection "perspective" "fov" 90

  #Position of shadowlight 1
  Rotate -153.4 1 0 0
  Rotate -90.0 0 0 1
  Translate 2.0 0.0 -4.0

WorldBegin
  ObjectInstance 1
  Rotate 360 1 1 0
  ObjectInstance 2
WorldEnd
MakeShadow "shadow1.pic" "shadowmap1.tx"
FrameEnd
FrameBegin
  Display "test1""file" "rgb"
  Format 400 320 1
  Projection "perspective" "fov" 40
  Translate 0 0 12
  Rotate -120 1 0 0
  Rotate 25 0 0 1
WorldBegin
  LightSource "shadowdistant" 1 "intensity" 2
  "from" [-2 0 4] "to" [0 0 0] "shadowname" "shadowmap1.tx"
  LightSource "ambientlight" 2 "intensity" 0.2
  Surface "plastic"
  Color 1 0 0
  ObjectInstance 1
  Color 1 1 0
  Rotate 360 1 1 0
  ObjectInstance 2
WorldEnd
FrameEnd
KeyFrameEnd
```



The lines printed in bold show the values that change during the animation. The cone tumbles through 360 degrees around an axis half way between the x and the y axes.