Shaping Up – library objects and polygons

Overview

This section explores the ways that objects are defined in a virtual world. Because our 3D worlds are described by hand written RIB files they will be relatively simple. However, this is not a disadvantage because it will focus attention on imparting as much visual interest through the use of careful shading techniques and sensitive lighting, rather than gratutious complexity obtained all too easily by the use of an interactive modelling system. Before embarking upon the intricacies of lighting and shading some competence must be gained with modelling. This section is designed to provide you with these skills.

"Shaping Up" takes an in-depth look at two types of surfaces commonly used to construct virtual models, namely, *quadrics* and *polygons*. Sophisticated modellers also use surfaces based on curves called splines. If you have used an illustration program such as Adobe Illustrator or Aldus FreeHand you would have employed 2D splines to create curves. However, 3D splines are an advanced topic of study and will not be addressed in this course.

Quadrics

Particular use will be made of the library of shapes, or primitives as they are sometimes called, that are built into RenderMan. These pre-defined surfaces are based upon mathematical expressions called quadratic equations, hence their general name of **quadric surfaces**. There are seven surfaces in the library and they are illustrated on the next two pages. Each quadric has its own set of parameters that allow its form to be accurately specified. The meaning of these parameters and examples of their use are given. In addition to being described by an equation they are also surfaces of revolution. That is, they are formed by spinning a line or curve around a central axis. Most modelling programs offer these primitives because it is easy to assemble them into composite models. Unlike many renderers RenderMan does not approximate quadrics in any way and so renders them with smooth silhouettes.

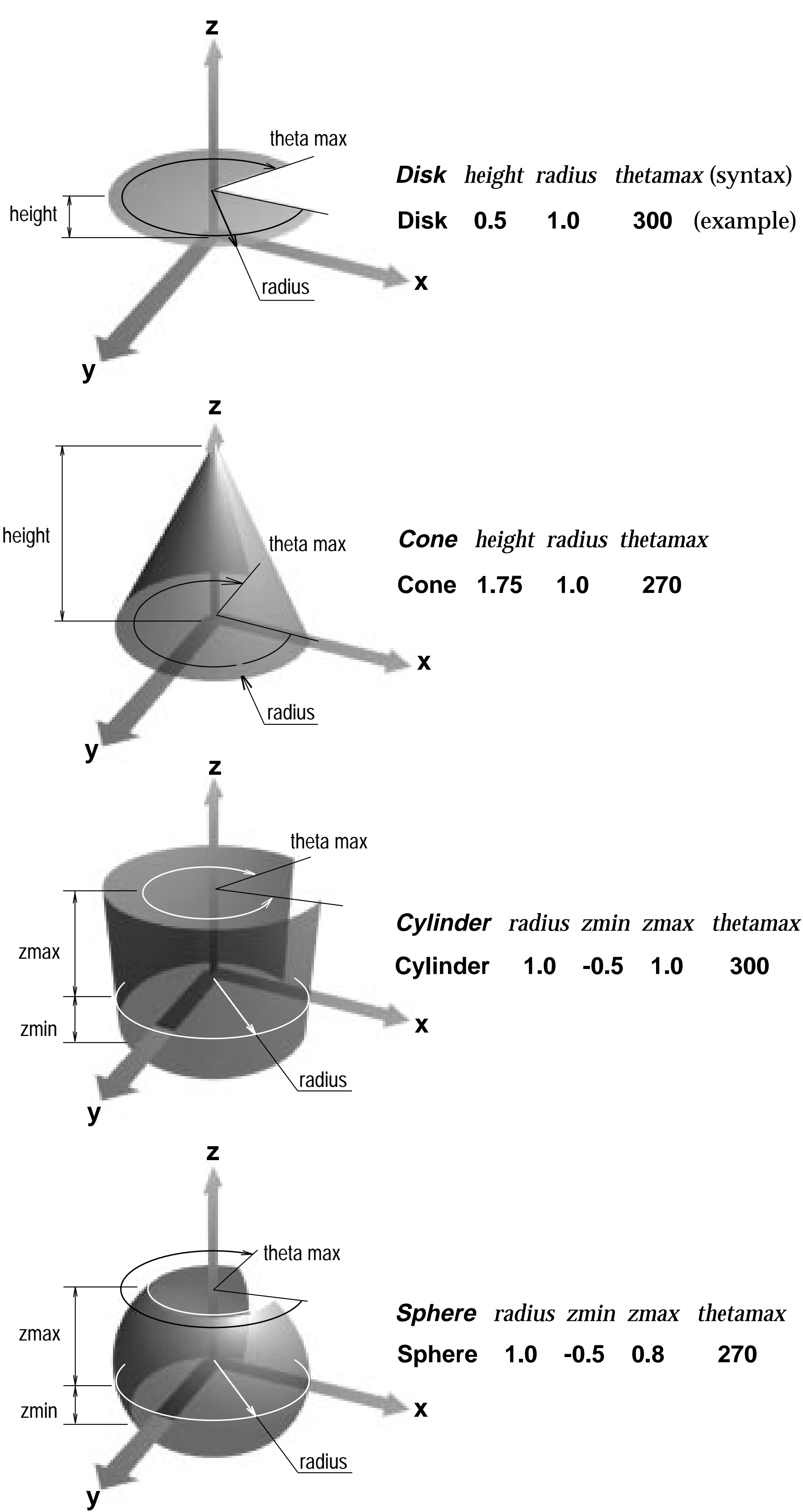
Polygons

The other type of surface that will be used is a polygon - a flat shape enclosed by straight edges. Traditionally, polygons have been very important in 3D computer graphics because of the ease with which they can be

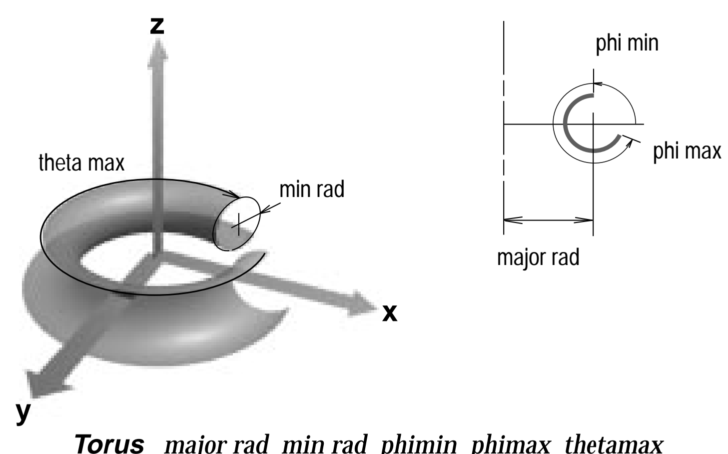
- internally represented by modellers and renderers,
- assembled into a skin or mesh that approximates a desired form, and
- rendered in a variety of ways to give the illusion of smoothness.

The straight edges of a polygon are defined by a sequence of 3D vertices each of which is specified by three numbers – its x, y and z coordinates. Since even simple polygon meshes can consist of dozens of polygons – each consisting of at least three vertices (ie. triangles), it will only be feasible for us to describe very simple surfaces.

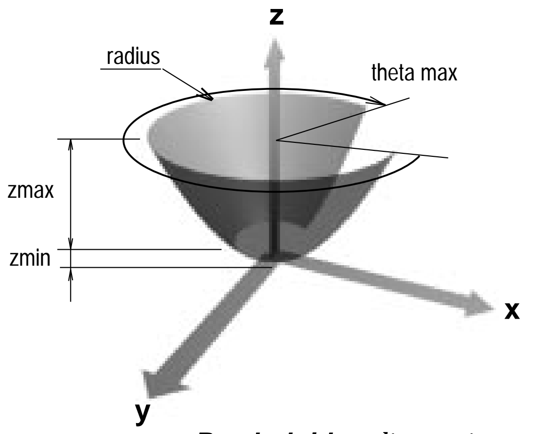
RenderMan's Library of Quadric Surfaces



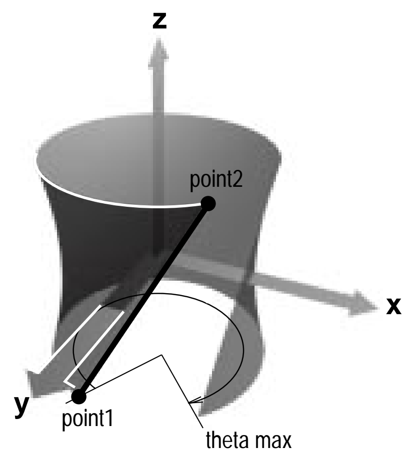
RenderMan's Library of Quadric Surfaces - continued



Torus	*major rad*	*min rad*	*phimin*	*phimax*	*thetamax*
Torus	**1.0**	**0.3**	**90**	**320**	**300**



Paraboloid	*radius*	*zmin*	*zmax*	*thetamax*
Paraboloid	**1.0**	**0.15**	**1.2**	**300**



Hyperboloid	*point1*	*point2*	*thetamax*
Hyperboloid	**-0.3 1.0 -1.0**	**0.7 0.7 1.0**	**300**

Example 1 - don't forget the inside!

RIB

```
#better goblet.RIB
#adding an inside surface
Display "goblet" "framebuffer" "rgb"
Projection "perspective" "fov" 40
Format 200 150 1

WorldBegin
   LightSource "pointlight" 1 "intensity" 40 "from" [4 2 4]
   Translate 0 0 5
   Rotate -120 1 0 0

   Surface "plastic"
   Color 1.0  0.9  0.3          #gold
   Cylinder 1 0 1.5 360         #container
   Disk 0 1 360                 #base of the container
   Cylinder 0.25 -1.5 0 360     #stem
   Disk -1.5 1 360              #base of the goblet

   Translate 0 0 1.5            #move the origin to the top of the goblet
   Sphere 1 -1 0 360            #hemi-spherical inside surface
WorldEnd
```

This example introduces the first of the library shapes – a sphere. It also uses two new RIB statements, LightSource and Surface. 3D computer graphics has developed a rich set of lighting and surface texturing techniques that can dramatically alter the appearance of an object. Although the concepts are dealt with in detail in later sections, light sources and material attributes can still be used effectively, even without elaborate explanations, to add realism to a model.

With the exception of those lines marked in italics, this file is the same as the final example of the previous section. At the end of the scene description the origin is moved to the top of the goblet and the lower half of a sphere is placed within the container by the RIB command,

```
Sphere   radius  zmin  zmax   thetamax
```

A (point) light source is oriented to high-light the curved surfaces of the goblet. The harshness of the lighting can be reduced by inserting this line,

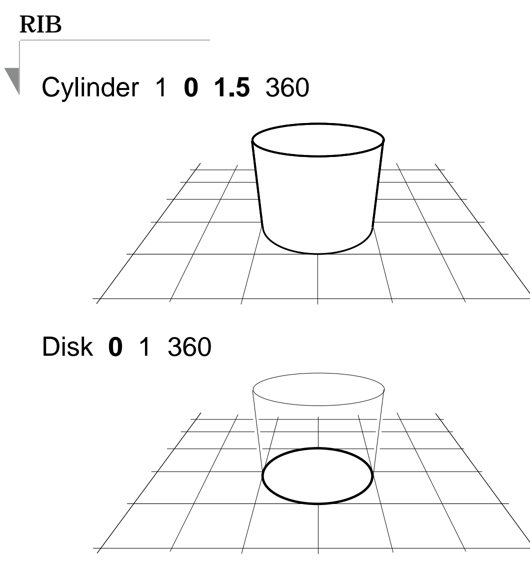
```
LightSource "ambientlight" 2 "intensity" 0.2
```

immediately after the first light source statement. The RIB command Surface, followed by the name of a material in the RenderMan library acts much like Color inthat all subsequent objects acquire the chosen characteristics. Although more will be said about materials and surface textures, you may like to experiment by substituting the parameter "plastic" for any one of those shown in the list given opposite. Later you will be shown how to control the characteristics of each material.

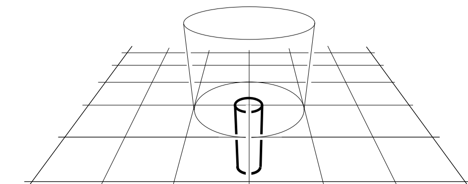
carpet
cloth
cmarble
constant
finemetal
Matte
metal
paintedplastic
plastic
rmarble
rsmetal
shinymetal
spatter
stone
wood

Placing objects in the world

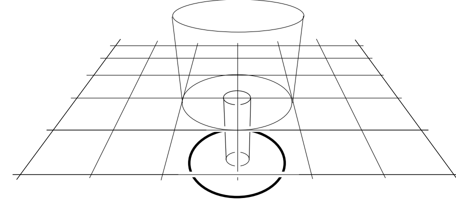
The actions of some of the RIB statements in the first example are illustrated below. In each case the position of the x–y plane is indicated by a grid. The surface being created is shown in the heavier line weight and the parameter(s) responsible for positioning the surface in the z direction are shown in bold.



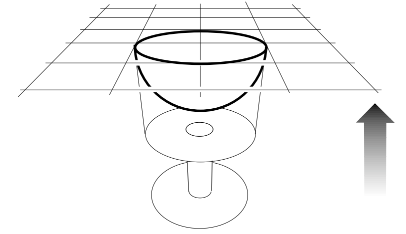
Cylinder 0.25 **-1.5** **0** 360



Disk **-1.5** 1 360



Translate 0 0 **1.5**
Sphere 1 **-1** **0** 360



Example 2 - adding a rim and moving the camera

RIB

```
#goblet with rim.RIB
#adding a rim
Display "goblet" "framebuffer" "rgb"
Projection "perspective" "fov" 40
Format 200 150 1

Translate 0 0 5
Rotate -120 1 0 0

WorldBegin
   LightSource "pointlight" 1 "intensity" 50 "from" [4 2 4]
   LightSource "ambientlight" 2 "intensity" 0.2

   Surface "plastic"
   Color 1.0  0.8  0.3          #gold
   Cylinder 1 0 1.5 360         #container
   Disk 0 1 360                 #base of the container
   Cylinder 0.25 -1.5 0 360     #stem
   Disk -1.5 1 360              #base of the goblet

   Translate 0 0 1.5            #move the origin to the top of the goblet
   Cylinder 0.9 -1.4 0 360
   Disk -1.4 0.9 360
   Torus 0.95 0.05 0 180 360
WorldEnd
```

Although this example demonstrates the use of a torus, its main feature is the way the transformations,

```
Translate 0 0 5
Rotate -120 1 0 0
```

that were previously used to rotate and move the goblet **within** the world space are now effecting the **whole** world. Remember, all RIB statements prior to WorldBegin refer to the way the world is oriented with respect to the camera. Because it makes more sense to change the camera–world relationship, as shown by the illustration on the next page, it will no longer be necessary to rotate and move individual objects to obtain a better view of them.

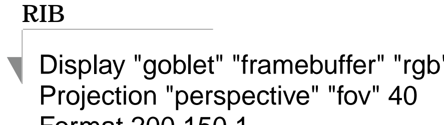
After a cylindrical liner and a flat base have been added to the inside of the goblet a rounded rim is created with the Torus statement,

```
Torus   major rad  min rad  phimin  phimax  thetamax
```

Try to add another disk to the base of the goblet and provide it with either a

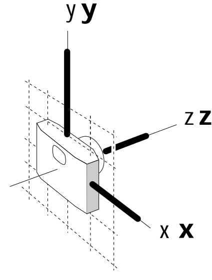
Positioning the world relative to the camera

RIB



```
Display "goblet" "framebuffer" "rgb"
Projection "perspective" "fov" 40
Format 200 150 1
```

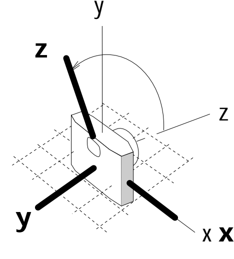
display the graphic in a window titled "goblet", 200 by 150 pixels in size, use a camera with a 40 degree field of vision and include rgb colour data



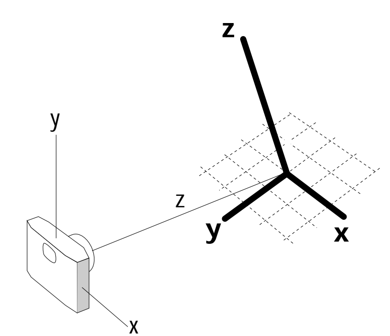
initially the origins of the camera and the world coincide

```
Translate  0  0  5
Rotate  -120  1  0  0
```

rotate the world -120 degrees around the x axis



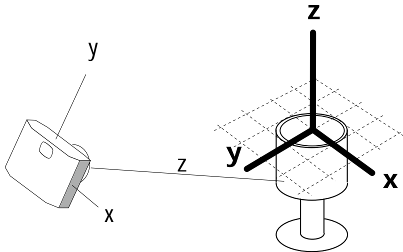
since transformations occur in reverse order the rotation is followed by the translation



move the world 5 units along the z axis of the camera

```
WorldBegin
   (assemble the goblet)
```

"freeze" the camera - now only use the world coordinates




```
WorldEnd
```

scene description complete

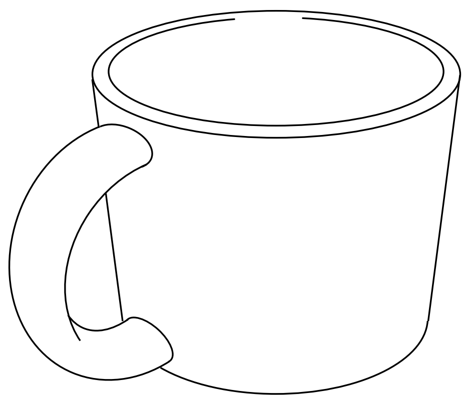
Example 3 - anyone for coffee?

RIB

```
#coffee mug.RIB
#modifying the goblet
Display "mug" "framebuffer" "rgb"
Projection "perspective" "fov" 40
Format 200 150 1

Translate 0 -0.5 5
Rotate -120 1 0 0
Rotate  45 0 0 1

WorldBegin
   LightSource "pointlight" 1 "intensity" 50 "from" [4 4 4]
   LightSource "ambientlight" 2 "intensity" 0.25

   Surface "plastic"
   Color 0 0 1                  #fully saturated blue
   Cylinder 1 0 1.5 360         #mug
   Disk 0 1 360                 #base of the mug
   Translate 0 0 1.5            #move the origin to the top
   Cylinder 0.9 -1.4 0 360      #lining of the mug
   Disk -1.4 0.9 360            # bottom of the mug
   Torus 0.95 0.05 0 180 360    #mug rim

   Translate 0 1 -0.75          #move the origin to the back, and lower it half way down the mug
   Rotate 90 0 1 0              #rotate the origin so that the handle will be vertical
   Torus 0.6  0.1  0 360 180    #create a handle
WorldEnd
```

In this example some minor alterations to the scene have changed the goblet into a coffee mug. The statements relating to the stem and base have been removed and those shown in bold have been added or altered. However, the most important point to notice about this file is the way the world is rotated 45 degrees clockwise about the z axis before it is tipped back 120 degrees. In all the previous examples the camera was vertically aligned with the y axis of the world. If you place a comment in front of the camera's second rotation you will immediately see the effect it has on the view. In addition, the mug has been 'centred' by moving the world 0.5 units down the y axis of the camera.

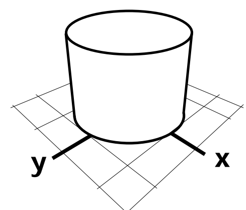
Introduce a Scale statement to widen the handle as shown. The mug does not look tall enough – increase its height to 1.9 units.



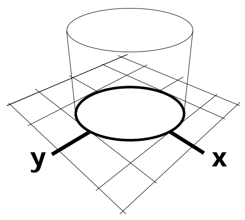
The actions of the RIB statements used in the construction of the coffee mug, example 3, are illustrated below. Unless otherwise indicated, the z axis is pointing up. The surface being created is shown in the heavier line weight and the parameter(s) responsible for positioning the suface in the z direction are shown in bold.

In the last diagram the handle has been widened by applying a scaling factor to the x coordinate. It is left as an exercise for you to determine where in the script the "Scale 2 1 1" command should be inserted.

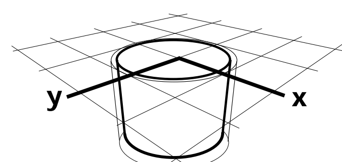
1 Cylinder 1 **0** **1.5** 360



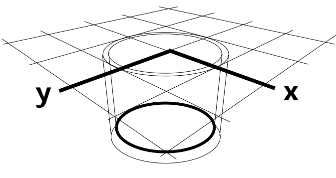
2 Disk **0** 1 360



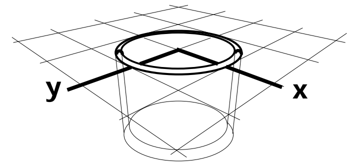
3 Translate 0 0 **1.5**
Cylinder 0.9 **-1.4** **0** 360



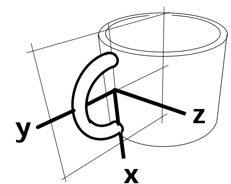
4 Disk **-1.4** 0.9 360



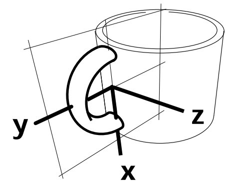
5 Torus 0.95 0.05 0 180 360



6 Translate 0 1 **-0.75**
Rotate 90 0 1 0
Torus 0.6 0.1 0 360 360



7 Scale 2 1 1



Question: why is the scaling being applied to the x axis when in this diagram it appears as if the z axis requires "stretching"?

Example 4 - the universal saucer

RIB

```
#saucer.RIB
#some tricky scaling
Display "saucer" "framebuffer" "rgb"
Projection "perspective" "fov" 40
Format 200 150 1

Translate 0 0 7
Rotate -120 1 0 0
Rotate 60 0 0 1

WorldBegin
  LightSource "pointlight" 1 "intensity" 45 "from" [2 -3 4]
  LightSource "ambientlight" 2 "intensity" 0.15

  Surface "plastic"
  Color 0.5 0.5 1      #pale blue

  Translate 0 0 0.5
  Scale 4.4 4.4 1

  Sphere 0.5 -0.5 0 360
  Sphere 0.4 -0.4 0 360
  Torus 0.45 0.05 0 360 360
WorldEnd
```

The purpose of this example is to show the effect of using scaling and translation, as well as the importance of applying these transformations in the correct order. Ignoring the transformations for a moment, the combination of the two spheres and the torus simply produces a hemi-spherical “cup”, 0.1 units in thickness, with a rounded rim.

The effect of the Scale statement is to stretch the “cup” into a saucer-like object, refer to the illustration on the next page. The translation is optional in that it does not change the form of the saucer, only its position. However, it makes sense to lift the saucer by an amount equal to its radius so that it ‘sits’ on the x–y plane, hence the translation of 0.5 units in the z direction.

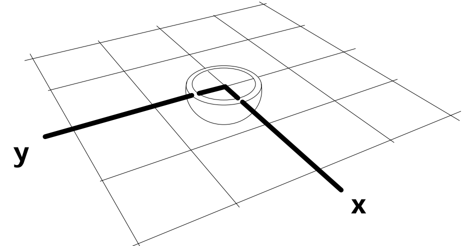
The widened rim of the saucer is due to the thickness of the basic “cup” being exaggerated by the scaling factor–like stretching a sheet of rubber. By adjusting the diameter of the inner sphere, and making the necessary changes to the parameters of the torus, a wide variety of rims can be created.

The basic “cup” can also be stretched vertically into an object reminiscent of an egg cup–see the next page. To create this object, x and y have been scaled by 1, therefore they remain unchanged, while the height in the z direction has been increased by 200%. To compensate for the scaling, the translation has been increased from 0.5 to 1 unit ie. 2 x 0.5 = 1.

As an exercise, create an egg by scaling a sphere, assign it an appropriate colour and position it in the egg cup.

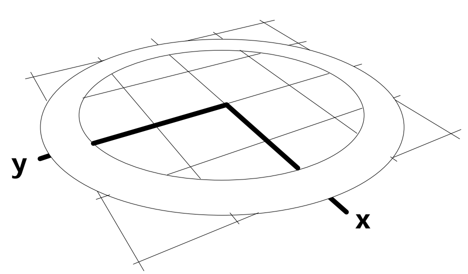
The effects of scaling and translation

no scaling or translation



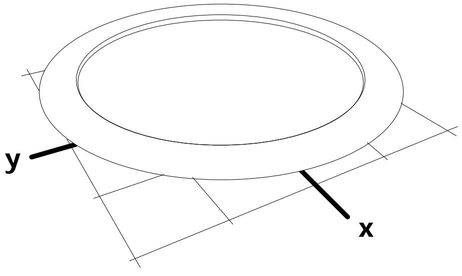

```
~~Translate 0 0 0.5~~
~~Scale 4.4 4.4 1~~

Sphere 0.5 -0.5 0 360
Sphere 0.4 -0.4 0 360
Torus 0.45 0.05 0 360 360
```

no translation



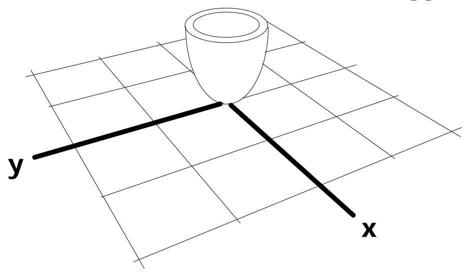

```
~~Translate 0 0 0.5~~
Scale 4.4 4.4 1

Sphere 0.5 -0.5 0 360
Sphere 0.4 -0.4 0 360
Torus 0.45 0.05 0 360 360
```

both scaling and translation applied




```
Translate 0 0 0.5
Scale 4.4 4.4 1

Sphere 0.5 -0.5 0 360
Sphere 0.4 -0.4 0 360
Torus 0.45 0.05 0 360 360
```

now its an egg cup!




```
Translate 0 0 1
Scale 1 1 2

Sphere 0.5 -0.5 0 360
Sphere 0.4 -0.4 0 360
Torus 0.45 0.05 0 360 360
```

For reference the original x–y plane BEFORE the transformations were applied are shown in each example.

Example 5 virtually “green”–reusable geometry

RIB

```
#eggcup with base.RIB
#copying and pasting with instancing
Display "eggcup" "framebuffer" "rgb"
Projection "perspective" "fov" 40
Format 200 150 1

ObjectBegin 1
   Sphere  0.5 -0.5 0 360
   Sphere  0.4 -0.4 0 360
   Torus  0.45 0.05 0 360 360
ObjectEnd

Translate 0 -.5 5
Rotate -120 1 0 0
Rotate 60 0 0 1

WorldBegin
   LightSource "pointlight" 1 "intensity" 25 "from" [2 -3 4]
   LightSource "ambientlight" 2 "intensity" 0.25

   #Egg cup top
   Color .55 .17 .11        #dark brown
   Surface "wood"
   Translate  0  0 1.0
   Scale  1  1  2
   ObjectInstance 1

   #Egg cup base
   Translate 0 0 -0.5
   Scale 1 1 0.25
   Rotate 180 1 0 0
   ObjectInstance 1
WorldEnd
```

The previous example illustrated an important point about 3D models; by making a few minor changes, to scaling for example, their geometry can form the basis of a variety of secondary models. A similiar principle can be applied within a single scene description. This example shows how several surfaces can be collected together into a single *retained* object, and conveniently reused, or *instanced*, many times. In the context of a drawing program this is like making a group, then copying and pasting it repeatedly within an illustration.

The intention to make an object from a collection of surfaces is indicated to the renderer by ObjectBegin/ObjectEnd. The number following ObjectBegin indentifies, or tags the collection for later use by a statement that places the object in the world, the number itself has no other significance,

ObjectInstance tag

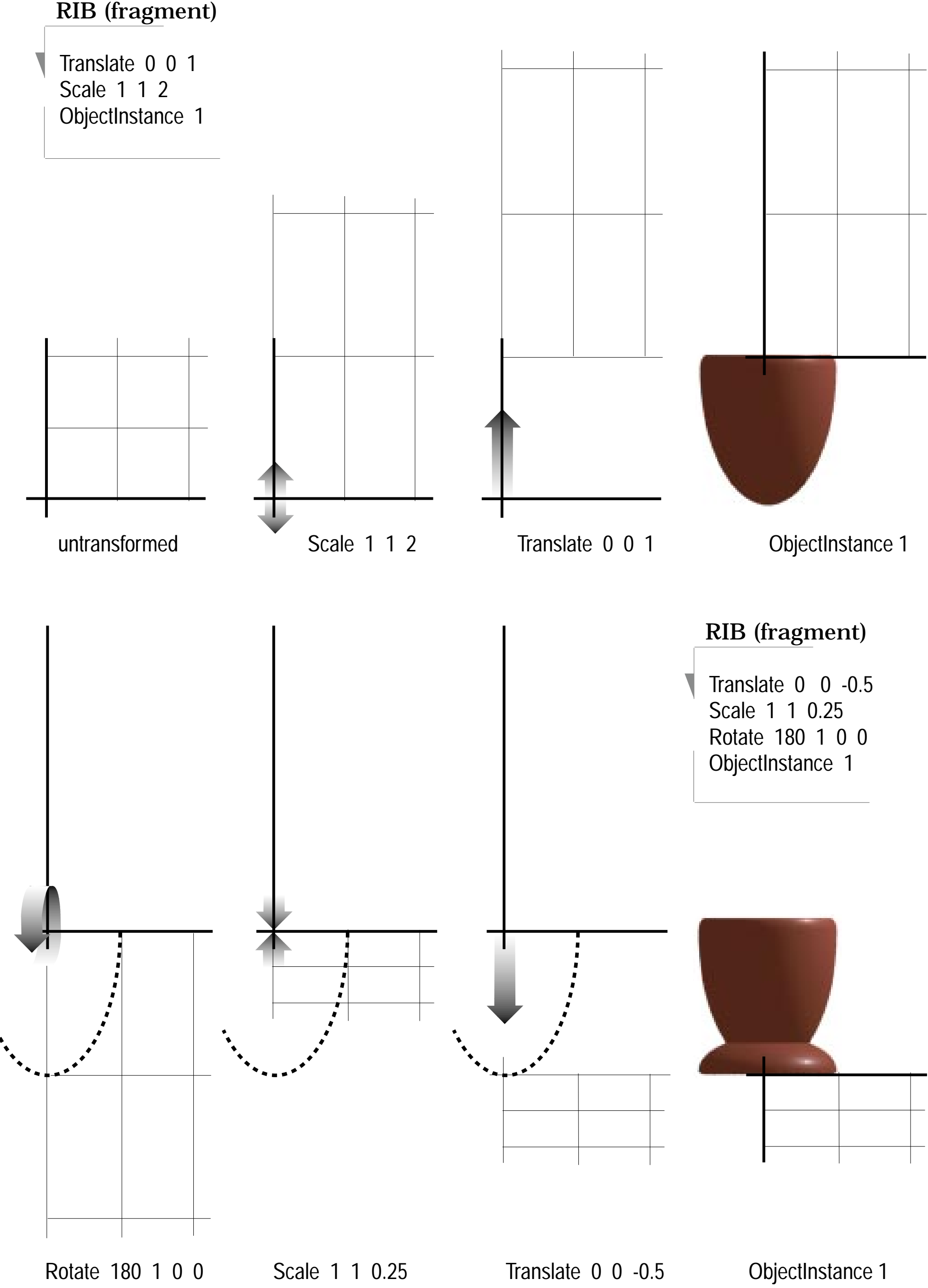
Unfortunately, transformations cannot be used between ObjectBegin and ObjectEnd. Instancing allows the renderer to work more efficiently and also helps to avoid writing tediously long RIB files. Pay particular attention to the next two pages as they explain the transformations used in this scene.

Visualising example 5

The following line drawings show the effect of applying the transformations used in example 5. At each stage, the coordinate system is represented by a one unit grid, subdivided into quarters. To fully understand the action of each group of transformations, remember they are applied,

- in reverse sequence, and
- with reference to the current coordinate system–shown as heavier lines.

The 'new' coordinate system **only** becomes current when an object is created.



Visualising example 5 – continued

In the last example, the base of the egg cup was positioned and placed in the scene by a scaling followed by a translation.

The same effect can also be achieved by placing the translation before the scaling. However, simply reversing the two statements will not work. As the drawing below shows, the translation must be altered. In general it is better to perform a scaling BEFORE a translation, as shown on the previous page.

RIB (fragment)

```
#Egg cup base
Scale  1  1  0.25
Translate  0  0  -2
Rotate  180  1  0  0
ObjectInstance 1
```

Rotate 180 1 0 0

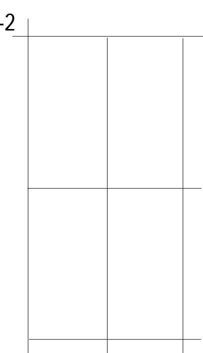
Translate 0 0 -2

Scale 1 1 0.25

ObjectInstance 1

-1

-2



Example 6 – playing with materials

RIB

```
#egg and cup.RIB
#playing with materials

Display "eggncup.tiff" "framebuffer" "rgb"
Projection "perspective" "fov" 40
Format 200 150 1
ShadingRate 5

ObjectBegin 1
   Sphere  0.5 -0.5 0 360
   Sphere  0.4 -0.4 0 360
   Torus  0.45 0.05 0 360 360
ObjectEnd

Translate 0 -0.7 4
Rotate -120 1 0 0
Rotate 60 0 0 1

WorldBegin
   LightSource "pointlight" 1 "intensity" 20 "from" [2 -3 4]
   LightSource "pointlight" 1 "intensity" 8 "from" [2 3 2]
   LightSource "ambientlight" 2 "intensity" 0.15

   #Table cloth
   Color 0.87  0.71  0.51
   Surface "carpet" "Kd" 1 "nap" 0.5 "scuff" 0.5
   Disk  0  20  360

   #Top
   Color  0.55  0.17  0.11
   Surface "wood" "darkcolor" [0 0 0] "swirl" 0.25 "grain" 15 "swirlfreq" 1.5
   Translate  0  0  1.0
   Scale  1  1  2
   ObjectInstance 1

   #Egg
   Surface "spatter" "basecolor" [0.87  0.66  0.6] "sizes" 3 "spattercolor" [0.55  0.17  0.11]
   "Ks" 0.0  "Kd" 1
   Sphere 0.4 -0.4  0.4 360

   #Base
   Color  0.55  0.17  0.11
   Surface "wood" "darkcolor" [0 0 0] "swirl" 0.25 "grain" 15 "swirlfreq" 1.5
   Translate  0  0  -0.5
   Scale  1  1  0.25
   Rotate  180  1  0  0
   ObjectInstance 1
WorldEnd
```

With the exception of a sphere that has been added to model an egg, this example is essentially the same as the previous scene. Where it is different, however, is not so much in the area of "shape" as "shading". Although the first example in this section experimented with various materials, this example exploits the way most *shaders* assigned with the Surface statement can have their properties 'tuned' by a number of parameters. Each parameter has a default value that can be either accepted, which is exactly what

happened in example 1, or reset as shown. Three *surface shaders* are used in this example to give the effect of a “carpet”, “wood” and “spatter”. Their full specifications, as documented by PIXAR, appear on the next three pages. Although the details of surface shading will be dealt with in another section, the descriptions of each of the materials used here should give you enough information to undertake your own experiments. Most of the parameters use values that range from 0 to 1; the exceptions, at least for the materials used in this example, are

wood "grain"
carpet "nap"
spatter "sizes"

Experiment with some or all the parameters in order to appreciate the control that each provides over the appearance of a surface. In the absence of pre-computed images this is a “trial and error” process. Three techniques can be used to speed up rendering

- use a higher ShadingRate, say 20, and optionally use
- ShadingInterpolation “smooth”, to reduce the blotchiness of the image,
- comment-out any surfaces that are not currently being adjusted.

ShadingRate is like a quality control adjuster; low values of around 1 or 2 give excellent results while higher values like 20 or more provide “rough and ready” snapshots. A high shading rate simply tells the renderer not to calculate the colour value for every pixel but to *sample* the pixels at whatever rate has been set. The closest comparison to ‘real world’ photography is choosing a high speed film with a coarse grain emulsion. Unfortunately, high shading rates generate very pixelated images, see opposite. To reduce these artefacts,

ShadingInterpolation "smooth"

can be used to tell the renderer to average-out, or interpolate, the pixels between the samples, otherwise it simply uses a constant colour. The statement can be inserted immediately after ShadingRate. Of course you may wish to take advantage of these image “defects” to achieve a particular illustrative effect, in which case resetting ShadingInterpolation is optional. By default its set to "constant", hence the blocks of flat colour.

Bearing in mind a PAL resolution video image constists of 442,368 pixels, the careful use of these statements can have a very significant effect on the speed of rendering.

Name **"wood" "Ka" "Ks" "Kd" "roughness" "specularcolor" "grain" "swirl" "swirlfreq" "c0" "c1" "darkcolor"**

Defaults "Ka" 1 "Ks" 0.4 "Kd" 0.6 "roughness" 0.2
"grain" 5 "swirl" 0.25 "swirlfreq" 1
"specularcolor" [1 1 1]
"darkcolor" [dependent on the surface colour]
"c0" [0 0 0] "c1" [0 0 1]

Description This shader creates a realistic-looking wood. The frequency of the wood grain can be changed with the grain parameter. The relative amount or amplitude of the turbuent swirl in the grain is controlled by the swirl paramater, and swirfreq controls the frequency of this turbulence. Low values of swirl produce more uniform looking wood, while low values of swirlfreq make the wood appear to be more knotty. Obviously these two parameters interact to a large extent. You should be careful not to set swirl too high or swirlfreq too low or the wood will become a jumbled mess.

The wood is simulated by creating a grain that is essentially composed of differently coloured concentric "cylinders" around a central axis defined by the two points c0 and c1. This axis is the z axis by default. Note that the orientation of this axis can be varied either by changing these two parameters or by doing some transformations between the call to the shader and the definition of the geometry. Either one of these approaches may make more intuitive sense in different applications.

The colour of the wood will normally consist of bands of different intensities of the surface colour. This is the most generally useful way of invoking the shader. However, for special appearances this can be changed by changing the darkcolor parameter, which controls the colour of the dark grain of the wood. The different intensity levels are actually levels of mixing between this colour and the surface colour, so setting the surface colour to red and darkcolor to white will produce red wood with white grain and various shades in between.

The parameters Ka, Ks and Kd have the usual meanings of ambient, specular and diffuse reflective intensities, respectively. roughness and specularcolor control the sharpness and colour of the specular highlight.

Bugs This shader can have problems with aliasing.

Name **"carpet" "Ka" "Kd" "scuff" "nap"**

Defaults "Ka" 0.1 "Kd" 0.6 "scuff" 1 "nap" 5 "swirl" 1

Description This shader produces a carpeted surface, complete with scuff-marks. The scuff parameter controls the "amount of scuff", or the relative frequency of intensity variations. Higher values produce more frequent scuffing. nap describes the "shagginess" of the carpet. Higher values make a more coarse-looking carpet.

The carpet shader makes a reasonable stab at anti-aliasing, so the actual grain of the carpet fades away with distance.

There are no specular reflections from real carpet (at least on a macroscopic scale), so the only lighting parameters are Ka and Kd, which have the usual meanings of ambient and diffuse reflective intensities, respectively.

Bugs The way anti-aliasing is performed can cause linear artifacts in some cases.

Name

"spatter" "Ka" "Ks" "Kd" "roughness" "specularcolor" "basecolor" "spattercolor" "specksize" "sizes"

Defaults

"Ka" 1 "Ks" 0.7 "Kd" 0.5 "roughness" 0.2
"specularcolor" [1 1 1] "basecolor" [0.1 0.1 0.5] "spattercolor" [1 1 1]
"specksize" 0.01 "sizes" 5

Description

This shader makes objects look like blue camp cookware with white paint spatters. Actually, both the blue basecolor and the white spattercolor can be changed if you desire.

The parameter specksize controls the size of the paint specks as you would expect. However, there are a range of sizes of paint specks controlled by the parameter sizes. Lower (integer) values produce smaller and more uniform specks. Higher values produce some larger blotches and specks of many different sizes.

The parameters Ka, Ks and Kd, have the usual meanings of ambient, specular and diffuse reflective intensities, respectively. roughness and specularcolor control the sharpness and colour of the specular highlight.

Bugs

This shader can have problems with aliasing.

Example 7a - making a composition the wrong way!

RIB

```
#egg and cup.RIB
#playing with materials

Display "eggncup.tiff" "framebuffer" "rgb"
Projection "perspective" "fov" 40
Format 200 150 1
ShadingRate 5

ObjectBegin 1
   Sphere  0.5 -0.5 0 360
   Sphere  0.4 -0.4 0 360
   Torus  0.45 0.05 0 360 360
ObjectEnd

Translate 0 -0.7 4
Rotate -120 1 0 0
Rotate 60 0 0 1

WorldBegin
   LightSource "pointlight" 1 "intensity" 20 "from" [2 -3 4]
   LightSource "pointlight" 1 "intensity" 8 "from" [2 3 2]
   LightSource "ambientlight" 2 "intensity" 0.15

   Surface "plastic"
   Color  0.5  0.5  1      #pale blue
   Translate  0 0 0.5
   Scale  4.4  4.4  1

   Sphere  0.5 -0.5 0 360
   Sphere  0.4 -0.4 0 360
   Torus  0.45 0.05 0 360 360

   #Top
   Color  0.55  0.17  0.11
   Surface "wood" "darkcolor" [0 0 0] "swirl" 0.25 "grain" 15 "swirlfreq" 1.5
   Translate  0  0  1.0
   Scale  1  1  2
   ObjectInstance 1

   #Egg
   Surface "spatter" "basecolor" [0.87  0.66  0.6] "sizes" 3 "spattercolor" [0.55  0.17  0.11]
   "Ks" 0.0  "Kd" 1
   Sphere 0.4 -0.4  0.4 360

   #Base
   Color  0.55  0.17  0.11
   Surface "wood" "darkcolor" [0 0 0] "swirl" 0.25 "grain" 15 "swirlfreq" 1.5
   Translate  0  0  -0.5
   Scale  1  1  0.25
   Rotate  180  1  0  0
   ObjectInstance 1
WorldEnd
```

In this example the description of the saucer in example 4, shown in bold, has been copied and pasted into the part of the previous RIB file that described the so-called table cloth. However, as the illustration on the next page shows, something very strange has happened to the egg cup and egg.

Because the saucer is created with a scaled coordinate system all the surfaces defined after this transformation are likewise effected, hence the Ostrich egg effect shown on the left, rather than the desired composition shown on the right!

Clearly objects in a scene need to have their individual coordinate systems, or *object space*, and their surface attributes kept, in a sense, private from each other. In a RIB file there are two ways in which this can be achieved. In the following example the two principle objects, the saucer and the egg cup holding an egg, are blocked together between the statements AttributeBegin/ AttributeEnd; these instruct RenderMan to localise (keep private) the geometry **AND** the surface attributes of each object. If only the geometry needs to be kept private, and there are good reasons why this is sometimes necessary, then the TransformBegin/TransformEnd statements are used instead.

To draw an analogy, if WorldBegin/WorldEnd define the beginning and end of an entire theatrical play, then the AttributeBegin/AttributeEnd behave like markers that separate one “scene” from another. Surfaces and polygons fulfill the role of “actors” with each, either separately or in collections, being assigned “costumes” represented by the attributes of Color and Surface.

In the improved RIB script on the following page, ObjectInstance has been used to insert the saucer instead of declaring three separate surfaces. None-theless, the composition still displays a modelling error–notice how the egg cup partially penetrates the saucer. However, because they are grouped together with the AttributeBegin/AttributeEnd statements, the egg cup and egg can be raised by a single transformation (shown in bold print on the next page). Because of the curvature of the saucer a similiar error accurs if the egg cup is moved toward the rim. But even so, it is sometimes acceptable to allow objects to interpenetrate as long as the error is not too noticable.

The two methods of grouping objects together using AttributeBegin/End and TransformBegin/End are summarised on page 24.

Example 7b - making a composition the correct way

RIB

```
#egg cup and saucer.RIB
#combining objects the correct way!

Display "eggncup.tiff" "framebuffer" "rgb"
Projection "perspective" "fov" 40
Format 200 150 1
ShadingRate 5

ObjectBegin 1
   Sphere  0.5 -0.5 0 360
   Sphere  0.4 -0.4 0 360
   Torus  0.45 0.05 0 360 360
ObjectEnd

Translate 0 -0.7 4
Rotate -120 1 0 0
Rotate 60 0 0 1

WorldBegin
   LightSource "pointlight" 1 "intensity" 20 "from" [2 -3 4]
   LightSource "pointlight" 1 "intensity" 8 "from" [2 3 2]
   LightSource "ambientlight" 2 "intensity" 0.15

   AttributeBegin    #Saucer
      Surface "plastic"
      Color .5 .5 1
      Translate 0 0 0.5
      Scale 4.4 4.4 1
      ObjectInstance 1
   AttributeEnd

   Translate 0 0 0.1    #raise the egg cup and egg

   AttributeBegin    #Egg cup and egg
      Color  0.55  0.17  0.11
      Surface "wood" "darkcolor" [0 0 0] "swirl" 0.25 "grain" 15 "swirlfreq" 1.5
      Translate  0  0  1.0
      Scale  1  1  2
      ObjectInstance 1

      #Egg
      Surface "spatter" "basecolor" [0.87  0.66  0.6] "sizes" 3 "spattercolor" [0.55  0.17  0.11]
      "Ks" 0.0  "Kd" 1
       Sphere 0.4 -0.4  0.4 360

      #Base
      Color  0.55  0.17  0.11
      Surface "wood" "darkcolor" [0 0 0] "swirl" 0.25 "grain" 15 "swirlfreq" 1.5
      Translate  0  0  -0.5
      Scale  1  1  0.25
      Rotate  180  1  0  0
      ObjectInstance 1
   AttributeEnd
WorldEnd
```

Example 7c - another way of grouping objects

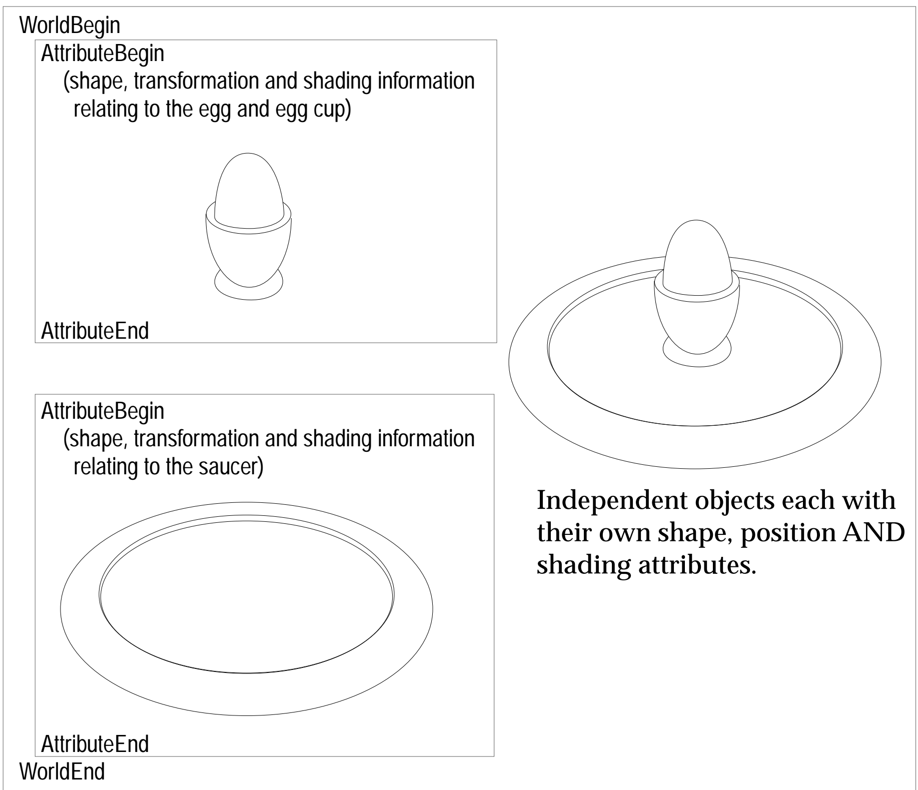
RIB fragment

```
WorldBegin
   #lighting setup the same as before...

   Color 0.87  0.71  0.51
   Surface "carpet" "Kd" 0.8 "nap" 0.8 "scuff" 0.8
   TransformBegin      #mug
      Cylinder 1 0 1.5 360
      Disk 0 1 360
      Translate 0 0 1.5
      Cylinder 0.9 -1.4 0 360
      Disk -1.4 0.9 360
      Torus 0.95 0.05 0 180 360

      #mug handle
      Scale 2 1 1
      Translate 0 1 -0.75
      Rotate 90 0 1 0
      Torus 0.6  0.1  0 360 180
   TransformEnd

   Translate 0 0 -0.15    #lower the saucer

   Color .65 .27 .21
   Surface "wood" "darkcolor" [0 0 0] "swirl" .25 "grain" 15 "swirlfreq" 1.5
   TransformBegin      #Saucer
      Translate 0 0 0.5
      Scale 4.4 4.4 1
      ObjectInstance 1
   TransformEnd
WorldEnd
```

This example is a combination of “coffee mug.RIB” and “saucer.RIB”. It shows how the surfaces and transformations that form an object can be grouped together using TransformBegin/TransformEnd. Like the groups formed by AttributeBegin/AttributeEnd in the previous example, these new statements keep the transformations for each object private, but they do so without localizing colour and surface attributes. The image on the left shows what happens if the transformations previously applied to the handle are not kept “private”. The image on the right is the result of the RIB given above.

Summary of methods relating to the grouping of objects

AttributeBegin
AttributeEnd



TransformBegin
TransformEnd