Computer Science 110
The Art & Science of Computer Graphics
Mount Holyoke College
Spring 2010

Topic Notes: Defining Custom Objects

Our primitive Mead objects can only get us so far. At some point, you will want something beyond spheres, cubes, cones, cylinders, and transformations of them. We will consider a number of new mechanisms that will allow us to build more complex objects with Mead.

Mesh Objects

We will look next at a way to define our own objects out of polygons – a *mesh* object.

We begin with an example – our own custom definition of a cube:

See Example:
`/home/jteresco/shared/cs110/examples/MeshCube`

In this example, we define a list of points that form one of the sides of our cube:

```
(define front
  '((-50 -50 -50) (50 -50 -50)
    (50 50 -50)   (-50 50 -50)))
```

These are the four corners of the "front" of the cube.

Now, anywhere from here on in our program, the name `front` can be used when we want to have that list of 4 points.

This is an example of a *polygon* and will come up in many of our custom object mechanisms.

For our purposes, the points of our polygon should be *co-planar*.

We next define the box out of 6 polygons:

```
(object box Mesh
        (addPoly front)
        (addPoly (polyXform front (xRot 90))) ; top
        (addPoly (polyXform front (xRot 180))) ; back
        (addPoly (polyXform front (xRot -90))) ; bottom
        (addPoly (polyXform front (yRot 90))) ; left
        (addPoly (polyXform front (yRot -90))) ; right
        )
```

We can now define objects other than those we could already get, such as a pyramid:

See Example:
`/home/jteresco/shared/cs110/examples/Pyramid`

Predefined Meshes

Mead includes a few predefined `Mesh` objects that you might find interesting:

- The Newell Teapot
- The Stanford Bunny

These are standard reference objects in computer graphics. Teapots show up in unexpected places in computer-generated animated films, for example.

A simple model that includes Mead's definitions of these:

See Example:
`/home/jteresco/shared/cs110/examples/TeapotAndBunny`

Prisms

Our next custom object mechanism is called a `Prism` object.

As with the mesh, we begin with a polygon. However, here, we are going to define a polygon in the xy-plane – all of the z coordinates are going to be 0.

For example, we can define a square with its corners aligned on the x- and y-axes:

```
(define square
        '((50 0 0) (0 50 0) (-50 0 0) (0 -50 0)))
```

This is a common enough occurrence that Mead provides a shortcut that lets us specify only the x- and y-coordinates and automatically adds the 0's in the z:

```
(define square
  (2to3d '((50 0) (0 50) (-50 0) (0 -50))))
```

This is a construct we will use frequently.

But the definition of a regular polygon with a given number of sides is common enough that Mead provides a further shortcut.

```
(polygon 4)
```

This computes the 4-sided regular polygon in the xy-plane and returns to us a list of those coordinates. We can use it in place of the hand-generated coordinate list from above:

```
(define square
    (2to3d (polygon 4)))
```

We can experiment with other regular polygons.

Armed with our polygon that represents a square, we can create our prism by “stretching out” or “extruding” the polygon along the z-axis.

```
(object squarePrism Prism
    (profile square)
)
```

Our square prism is an object we can add and transform like our other objects. It has a profile of the polygon provided and stretches from -50 to +50 in the z dimension.

Sweep

We can create mesh objects by “sweeping” a polygon as well.

We begin with our square in the xy-plane, defined as above.

We then transform it to be a small rectangle offset along the x-axis:

```
(define littleRectangle
  (polyXform square
             (compose
              (zRot 45)
              (scale 2 .1 1) ; z cannot be 0!
              (translate 200 0 0)
              )
             )
  )
```

We can then “sweep” this polygon about the y-axis:

```
(tell scene
      (add (sweep littleRectangle 12) redPlaster)
      )
```

The result of the `sweep` operation is a `Mesh` object consisting of 12 copies of the original polygon, evenly spaced in a circle about the y-axis (in this case, one every 30 degrees), with neighboring copies connected to each other by solid material.

If we increase the 12, the object begins to look more and more round, but the object is more and more complex, resulting in a more expensive rendering by Mead.

See Example:
`/home/jteresco/shared/cs110/examples/SweptPlatform`

Lathe Objects

While the sweep operation can approximate a round object, a `Lathe` object will produce a truly round object:

```
(object washer Lathe
        (profile littleRectangle))
```

If we look closely at an object created with the above, we see that it has no outside surface.

To include that surface, we need to repeat the first point in our polygon as the last point. As this is another very common opertion, Mead provides a function that takes a list of points and returns to us a new list of points with the first point repeated as the last:

```
(object washer Lathe
        (profile (closeList littleRectangle)))
```

See Example:
`/home/jteresco/shared/cs110/examples/Washer`

Constructive Solid Geometry

The term for the techniques we will examine is *Constructive Solid Geometry (CSG)*.

We bring two or more objects together to treat them as a unit.

We have already seen one CSG construct: the `Group` object. Each object added to a `Group` is accumulated or "unioned" together.

In addition to the groups we have created, the `scene` is a predefined `Group`.

`Intersection`s and `Difference`s work as you might expect:

- an object defined by the *intersection* of a set of objects is the portion that is in common to all objects
- an object defined by the *difference* of a set of objects is the portion of the first that is not also part of the others

Just using a few of our primitive objects, we can see how these work:

```
(object halfSphere Difference
        (add sphere)
        (add cube (translate 0 0 -50)))
```

The resulting object is the "back" half of the unit sphere (the part in negative z). The object specified by the second `add` is "subtracted" from the object specified by the first.

Note carefully that order matters for differences. If we reverse the order above, we get a cube with a half sphere scooped out.

For an intersection, order doesn't matter:

```
(object halfSphere Intersection
        (add sphere)
        (add cube (translate 0 0 -50)))
```

Here, we also get a half sphere, but only the half that is covered by the cube instead of the half that was not covered by the cube.

In each of these cases, we use the `add` message to include components in our CSG unit, even when the operation is more logically thought of as a "subtract" or "intersect" instead.

A simple example:

See Example:
`/home/jteresco/shared/cs110/examples/Arrows`

A much more complex example:

See Example:
`/home/jteresco/shared/cs110/examples/Lamp`

Note the use of names that we define in place of numbers. This is a good programming practice. When you have a number in your program that has a specific meaning in the context in which it's used, define a name for it and use that name instead. It will make your program easier to read and understand both for you and for others who might be trying to understand it.

Extruded Objects

Several people have asked about more general methods to make prism-like objects. Recall that the `Prism` object is restricted to taking a polygon in the xy-plane and stretching it from -50 to +50 in the z dimension.

We can define a polygon and "extrude" it to a three dimensional object by specifying a transformation. This is more flexible than the `Prism` object since we can start with any polygon and extrude using any transformation.

Here's one where we rotate and translate to get from one end to the other.

See Example:
`/home/jteresco/shared/cs110/examples/ExtrudePenta`

In our next example, we create an “hourglass” object by taking a 100-sided polygon as our start and rotate by a large amount during the extrude.

See Example:
`/home/jteresco/shared/cs110/examples/Hourglass`

This is a pretty flexible construct and we will see more interesting examples later.