More Object Representations

- 3D scanning
- Parametric instancing
- Hierarchical modeling
- Constructive Solid Geometry
- Sweep Objects
- Implicit Surfaces

Meshes from Scanning

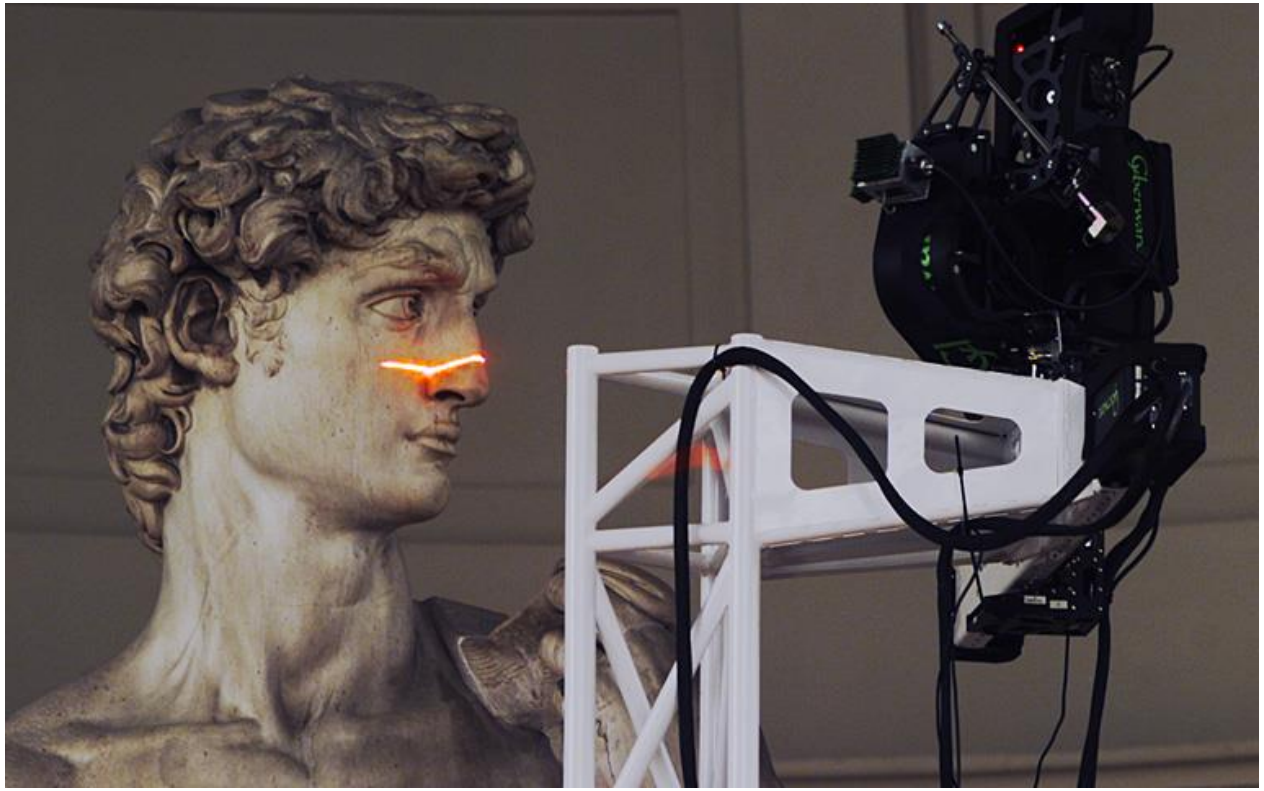
- Laser scanners sample 3D positions
  - One method uses triangulation
  - Another method uses time of flight
  - Some take images also for use as textures
  - Famous example: Scanning the David
- Software then takes thousands of points and builds a polygon mesh out of them
- Research topics:
  - Reduce the number of points in the mesh
  - Reconstruction and re-sampling!

3D Scanning Tools

Laser head scanning

Mechanical digitizing

Scanning in Action



http://www-graphics.stanford.edu/projects/mich/

3D Scanning Demo

- 3d scanning intro

http://www.youtube.com/watch?v=IQI0o2gxlok&feature=related

- 3d church

http://www.youtube.com/watch?v=ahwqCl5Z9oU&feature=related

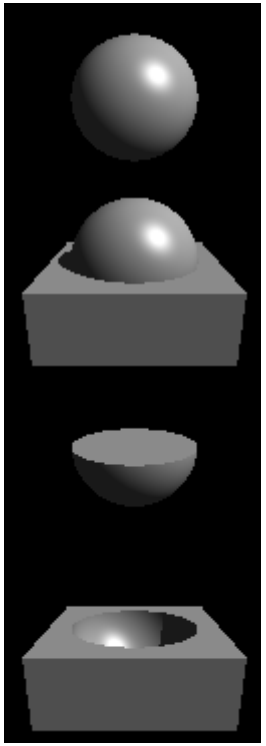
- handheld scanner

http://www.youtube.com/watch?v=2NeIJ-wJSVY

Constructive Solid Geometry (CSG)

- Based on a tree structure, like hierarchical modeling, but now:
  - The internal nodes are set operations: union, intersection or difference (sometimes complement)
  - The edges of the tree have transformations associated with them
  - The leaves contain only geometry
- Allows complex shapes with only a few primitives
  - Common primitives are cylinders, cubes, etc, or quadric surfaces
- Motivated by computer aided design and manufacture
  - *Difference* is like drilling or milling
  - A common format in CAD products

CSG Examples



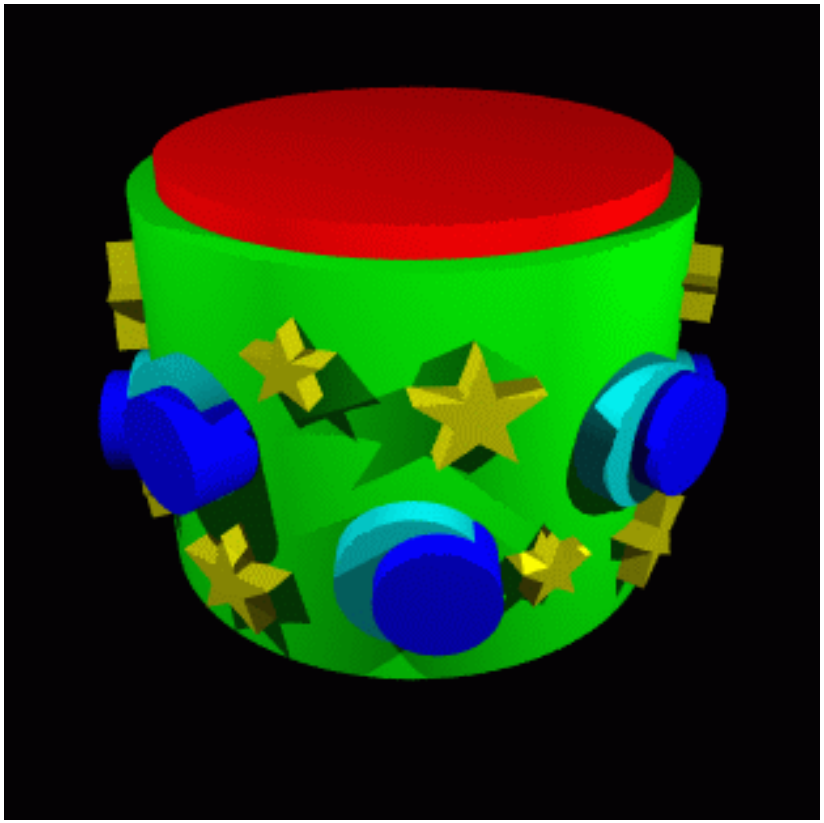
The two primitives: a sphere and a box.

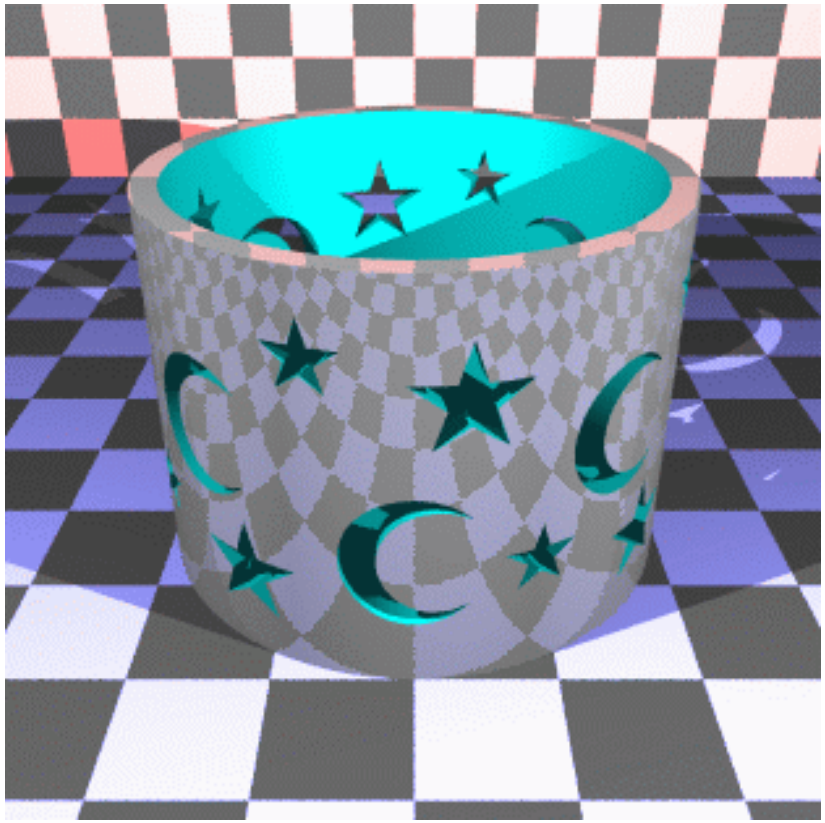
The union of the two primitives.

The intersection of the two primitives.

The difference of the two primitives: box minus sphere.

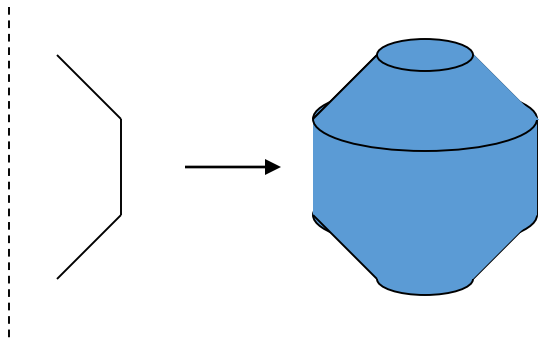
CSG Examples

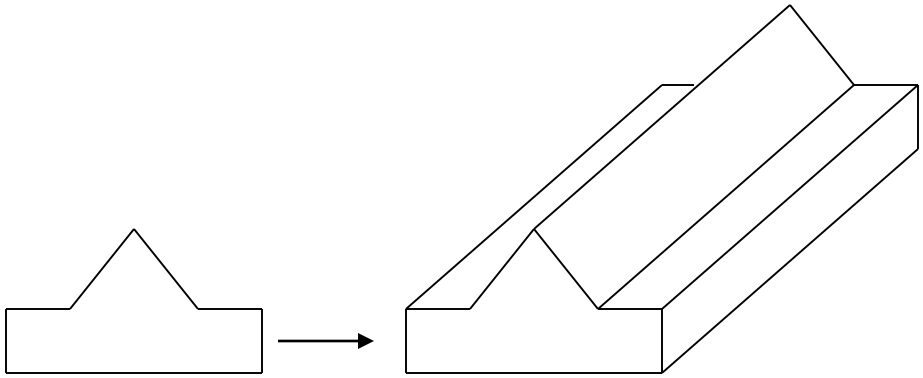




Sweep Objects

- Define a polygon by its edges
- Sweep it along a path
- The path taken by the edges form a surface - the sweep surface
- Special cases
  - Surface of revolution: Rotate edges about an axis
  - Extrusion: Sweep along a straight line





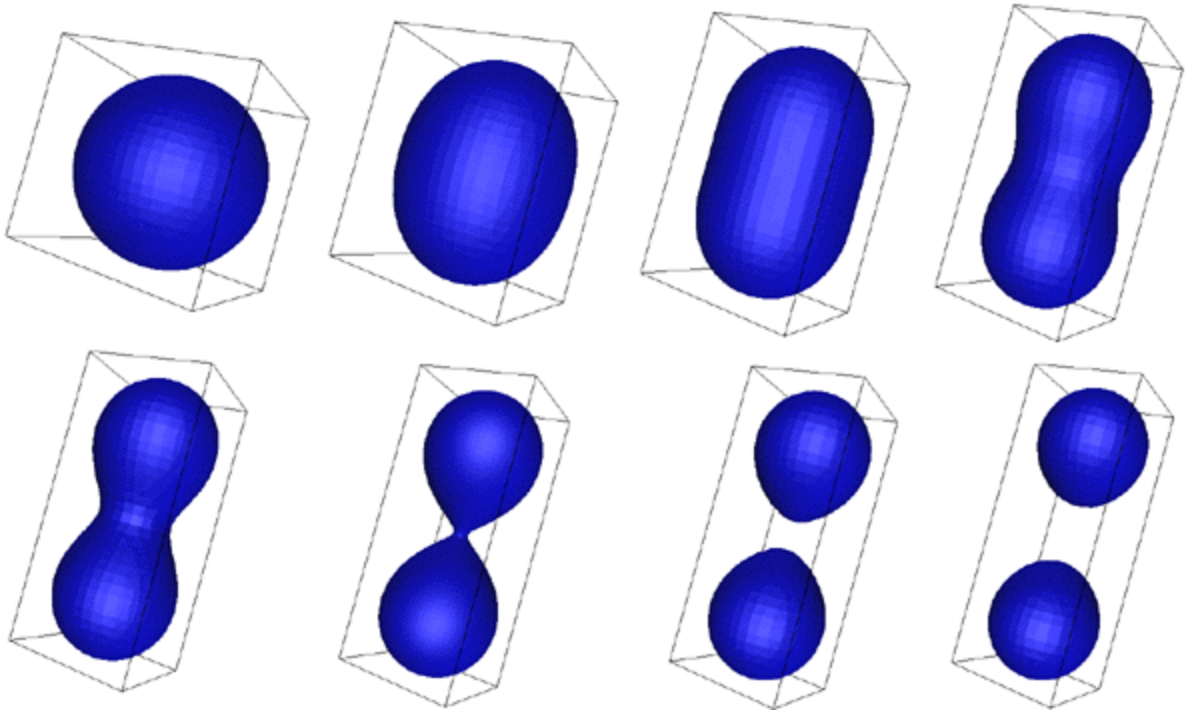
Implicit Functions

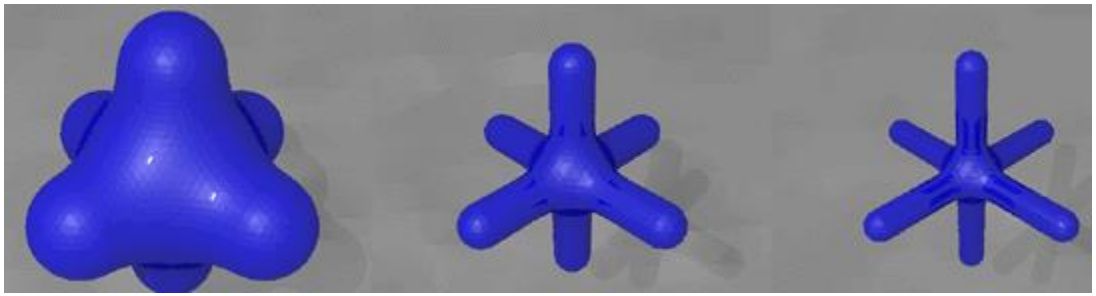
- Some surfaces can be represented as the vanishing points of functions (defined over 3D space)
  - Places where a function *f(x,y,z)=0*
- Some objects are easy represent this way
  - Spheres, ellipses, and similar
  - More generally, quadratic surfaces:

  - Shapes depends on all the parameters *a,b,c,d,e,f,g*

$$ax^2 + bx + cy^2 + dy + ez^2 + fz + g = 0$$

Implicit Surfaces

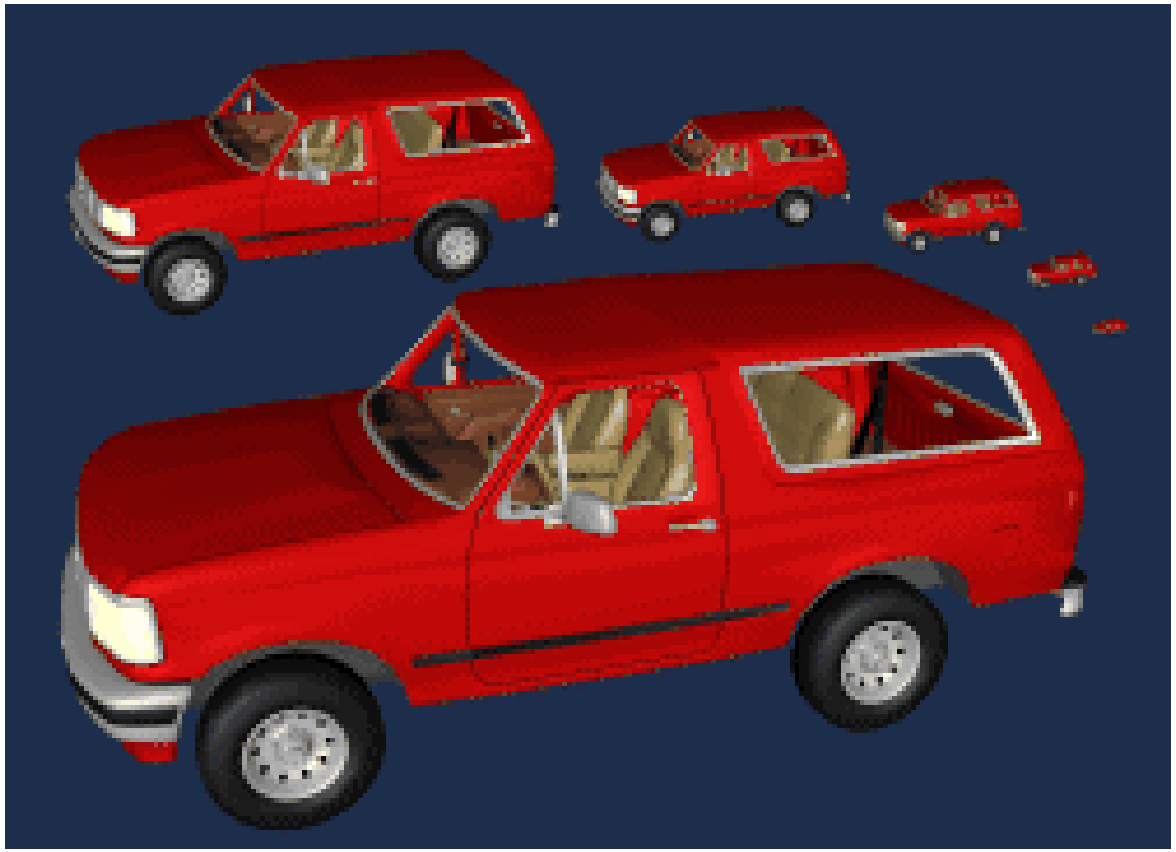




Level Of Detail

- There is no point in having more than 1 polygon per pixel
  - Or a few, if anti-aliasing
- Level of detail strategies attempt to balance the resolution of the mesh against the viewing conditions
  - Must have a way to reduce the complexity of meshes
  - Must have a way to switch from one mesh to another
  - An ongoing research topic, made even more important as laser scanning becomes popular
  - Also called mesh decimation, multi-resolution modeling and other things

Level of Detail



http://www.cs.unc.edu/~geom/SUCC_MAP/

Problems with Polygons

- They are inherently an approximation
  - Things like silhouettes can never be prefect without very large numbers of polygons, and corresponding expense
  - Normal vectors are not specified everywhere
- Interaction is a problem
  - Dragging points around is time consuming
  - Maintaining things like smoothness is difficult
- Low level representation
  - Eg: Hard to increase, or decrease, the resolution
  - Hard to extract information like curvature

Parametric Instancing

- Many things, called primitives, are conveniently described by a label and a few parameters
  - Cylinder: Radius, length, does it have end-caps, …
  - Bolts: length, diameter, thread pitch, …
  - Other examples?
- This is a modeling format:
  - Provide software that knows how to draw the object given the parameters, or knows how to produce a polygonal mesh
  - How you manage the model depends on the rendering style
  - Can be an exact representation

Rendering Instances

- Generally, provide a routine that takes the parameters and produces a polygonal representation
  - Conveniently brings parametric instancing into the rendering pipeline
  - May include texture maps, normal vectors, colors, etc
  - OpenGL utility library (GLu) defines routines for cubes, cylinders, disks, and other common shapes
  - Renderman does similar things, so does POVray, …
- The procedure may be dynamic
  - For example, adjust the polygon resolution according to distance from the viewer

Modeling Support in OpenGL
Display List

- What is Display List
- *Display list* is a group of OpenGL commands that have been stored for later execution.
- When a *display list* is invoked, the commands in it are executed in the order in which they were issued.
- Immediate mode
  - commands are executed immediately
- You can freely mix immediate mode programming and display lists.

Why Use Display Lists?

- It can reduce OpenGL command transmission
  - When running OpenGL programs remotely to another machine on the network. Display lists are stored in server computer. You can reduce the cost of repeatedly transmitting commands over the network.
- Performance Gain in Local Running
  - Some graphics hardware may store display lists in dedicated memory or may store the data in an optimized form that is more compatible with the graphics hardware or software.

Display Modes

- Immediate Mode. This is the OpenGL default. Commands are executed immediately.
- Retained Mode. Sometimes called deferred mode. Commands are stored for later use. Display lists provide this functionality in OpenGL.

Display List Advantages

- Only the results of calculations are stored, for example the final matrix associated with a sequence of transformations.
- A display list that represents an object such as the wheel of a car can be easily rendered multiple times in different positions by using a display list.
- Network performance is enhanced because commands are stored on the display server.
- Display lists can be used to implement level of detail.

Display List Disadvantages

- Display lists are for the speed at the cost of memory, so more memory will be consumed when using display lists.
- For small display lists, list overhead may negate any speed advantage.
- Speedup with display lists is implementation-dependent.

A Simple Example

```
void init(void) {
     GLuint theTorus = glGenLists(1);
     glNewList(theTorus, GL_COMPILE);
          torus(8, 25)
     glEndList();
}

void display(void) {
     glCallList(theTorus);
     glFlush();
}
```

A Simple Example (cont)

```
void torus(int numc, int numt) {
 for(i=0; i<numc; i++) {
  glBegin(GL_QUAD_STRIP);
  for(j=0; j<=numt; j++) {
   for(k=1; k>=0; k--) {
    x = (1+.1*cos(s(twopi/umc))*cos(t*twopi/numt);
    y = (1+.1*cos(s(twopi/umc))*sin(t*twopi/numt);
    z = .1 * sin(s*twopi/numc);
    }
   }
   glEnd();
 }
}
```

Best uses of Display Lists

- Matrix operations (e.g. transformations)
- Raster bitmaps and images
- Lighting and materials
- Textures
- Polygon Stippling
- State variables
- Nurbs
- Polygon Tesselation

Naming a Display List

- Each display list is identified by an integer index.

- To allocate one or more unused index, use **`glGenLists()`** command.
  - **`listIndex = glGenLists(n);`** allocates **`n`** consecutive previously unallocated display-list indices, and returns the first index.

Creating a Display List

- Use **`glNewList()`** and **`glEndList()`**

```
glNewList(listIndex, GL_COMPILE);
    ...
glEndList();
```

- Options for **`glNewList()`**
  - **`GL_COMPILE_AND_EXECUTE`**
  - **`GL_COMPILE`**

Creating a Display List

```
GLuint tri_list;
void MakeList()
{
    tri_list = glGenLists(1);
    // Note: can also use GL_COMPILE_AND_EXECUTE with glNewList
    glNewList(tri_list, GL_COMPILE);
        glColor3f(1.0, 0.0, 0.0);
        glBegin(GL_TRIANGLES);
            glVertex2f(0.0, 0.0);
            glVertex2f(1.0, 0.0);
            glVertex2f(0.0, 1.0);
        glEnd();
    glEndList();
}
```

Invoking a Display List

```
void display()
{
    for (int i = 0;  i < 10;  i ++)
    {
        glCallList(tri_list);
        glTranslatef(1.5, 0.0, 0.0);
    }
    glFlush();
}
```

Deleting a Display List

```
int main(int argc, char** argv)
{
.
.
.
    glutMainLoop();
    glDeleteLists(tri_list, 1);
    return 0;
}
```

Hierarchical Display Lists

```
void MakeList()
{
    glNewList(bicycle, GL_COMPILE);
        glCallList(handlebars);
        glCallList(frame);
        glTranslatef(1.0, 0.0, 0.0);
        glCallList(wheel);
        glTranslatef(3.0, 0.0, 0.0);
        glCallList(wheel);
    glEndList();
}
```

Hierarchical Display Lists (cont.)

```
// The following code is illegal!
glNewList(list1, GL_COMPILE);
    glBegin(GL_TRIANGLES);
    …
    glEnd();
    glNewList(list2, GL_COMPILE);  // Illegal! Use glCallList() instead.
        glBegin(GL_TRIANGLES);
        …
        glEnd();
    glEndList();
glEndList();
```

Managing State Variables

```
glNewList(poly_list, GL_COMPILE);
    glPushMatrix();
    glPushAttrib(GL_CURRENT_BIT);  // many options – see blue book
    glTranslatef(1.5, 0.0, 0.0);
    glColor3f(1.0, 0.0, 0.0);
    glBegin(GL_POLYGON);
    …
    glEnd();
    glPopAttrib();     // restores original color
    glPopMatrix();     // restores original position
glEndList();
```