OpenGL Picking Made Easy

Overview

In science and engineering 3D visualization applications, it is useful for the user to point to something and have the program figure out what is being pointed to. This is called *picking*.

You can imagine how you could do this in software. You would do all of the transformations yourself and figure out where each object ends up being drawn on the screen and then check its closeness to the cursor. The good news is that this can be done. The bad news is that none of us wants to do it, and it would be painfully slow if we did.

Mercifully, OpenGL allows the hardware to help us out. The strategy is simple:

1. **Tell the hardware that we are in picking ("selection") mode.**

2. **Ask the hardware to pretend it is drawing the scene. Go through all the motions, just don't actually color any pixels.**

3. **As part of drawing the scene, we occasionally give the hardware "names" for the objects we are drawing.**

4. **Whenever the hardware finds that it would have colored a pixel within a certain tolerance of the cursor, it returns to us the object "name" that it is currently holding. This tells you what set of drawn objects must have passed near the cursor.**

5. **Tell the hardware that we are back in drawing ("render") mode.**

Setup

Let's look at all the steps in detail. The first step is to setup some things in the #defines. This typically includes a picking tolerance (in pixels) and how large to make the array that the hardware will use to record the names that it finds upon a successful pick:

```
/* picking tolerance in pixels:     */
#define PICK_TOL              10.

/* how big to make the pick buffer: */
#define PICK_BUFFER_SIZE      256
```

Next, declare the picking buffer and a way to keep track of the rendering mode in the global variables:

```
unsigned int PickBuffer[PICK_BUFFER_SIZE];  /* picking buffer */
int          RenderMode;                    /* GL_RENDER or GL_SELECT */
```

setupOpengl()

In `setupOpenGL()`, tell the hardware what picking array to use and how big it is. You *must* do this *after* you create the window:

```
   /* setup the picking buffer: */
   glSelectBuffer( PICK_BUFFER_SIZE, PickBuffer );
```

Assigning Pick Names

Now comes the part that requires you to plan some strategy: assigning pick names. The names are really unsigned 32-bit integer numbers that get passed down to the hardware during display. The hardware will tell you what name(s) it is holding at the moment a pick occurs. Thus, use enough unique names to establish the identity of the individual objects. For example:

```
    glLoadName( 0 );
    CSCI441::drawWireSphere( 1.0, 15, 15 );
    glLoadName( 1 );
    CSCI441::drawWireCube( 1.5 );
    glLoadName( 2 );
    CSCI441::drawWireCone( 1.0, 1.5, 20, 20 );
    glLoadName( 3 );
    CSCI441::drawWireTorus( 0.5, 0.75, 20, 20 );
```

would work nicely to distinguish four shapes. Note that you can have as many graphics objects after a call to glLoadName() as you'd like. A pick on any of those objects, though, will return the same name.

One thing to be aware of, however, is that a call to `glLoadName()` cannot be embedded in a `glBegin()`-`glEnd()` pair. Thus, if you are displaying a collection of triangles, you ***cannot*** say:

```
glBegin( GL_TRIANGLES ); {
  for( unsigned int i = 0; i < NTRIS; i++ ) {
      glLoadName( i );
      glVertex3f( Tris[i].x0, Tris[i],y0, Tris[i].z0 );
      glVertex3f( Tris[i].x1, Tris[i],y1, Tris[i].z1 );
      glVertex3f( Tris[i].x2, Tris[i],y2, Tris[i].z2 );
  }
}; glEnd();
```

Instead, you ***must*** say:

```
for( unsigned int i = 0; i < NTRIS; i++ ) {
  glLoadName( i );
  glBegin( GL_TRIANGLES ); {
    glVertex3f( Tris[i].x0, Tris[i],y0, Tris[i].z0 );
    glVertex3f( Tris[i].x1, Tris[i],y1, Tris[i].z1 );
    glVertex3f( Tris[i].x2, Tris[i],y2, Tris[i].z2 );
  }; glEnd();
}
```

The pick names can be more sophisticated (i.e., more complicated) than just a single name. The name-holding place in the hardware is actually a stack, so multiple names can be pushed onto the stack. This is useful for hierarchical situations such as:

```
glLoadName( JAGUAR );
glPushName( BODY ); {
    glCallList( JagBodyList );
}; glPopName();

glPushName( FRONT_LEFT_TIRE ); {
    // translate
    glCallList( TireList );
}; glPopName();

glPushName( FRONT_RIGHT_TIRE ); {
    // translate
    glCallList( TireList );
}; glPopName();

glLoadName( YUGO );
glPushName( BODY ); {
    glCallList( YugoBodyList );
}; glPopName();
```

so that by looking at the 2 pick names that are returned, you know which car *and* what part of that car was picked.

Let's get ready to draw. Be sure that the program knows to startup in render mode, not picking mode. So,in `reset()`:

```
RenderMode = GL_RENDER;
```

Now, when the pick is triggered by a mouse button press:

1. **Set the mode to pick mode (`GL_SELECT`)**

2. **Call `renderScene()` to do the pretend drawing**

3. **Set the mode back to render mode (`GL_RENDER`)**

4. **Examine the picking array**

mouse_button_callback()

In `mouse_button_callback()`, do the following:

```
    if( (button == GLFW_MOUSE_BUTTON_LEFT) && (action == GLFW_PRESS) ) {
            RenderMode = GL_SELECT;
            glRenderMode( GL_SELECT );
            renderScene();
            RenderMode = GL_RENDER;
            Nhits = glRenderMode( GL_RENDER );
#ifdef BUG_KLUDGE
            if( Nhits == 0 ) {
                 RenderMode = GL_SELECT;
                 glRenderMode( GL_SELECT );
                 renderScene();
                 RenderMode = GL_RENDER;
                 Nhits = glRenderMode( GL_RENDER );
            }
#endif
            if( DEBUG )
                fprintf( stderr, "# pick hits = %d\n", Nhits );
            for( int i = 0, index = 0; i < Nhits; i++ ) {
                nitems = PickBuffer[index++];
                zmin = PickBuffer[index++];
                zmax = PickBuffer[index++];
                if( DEBUG ) {
                    fprintf( stderr, "Hit # %2d: found %2d items on the
                             name stack\n", i, nitems );
                    fprintf( stderr, "\tZmin = 0x%0x, Zmax = 0x%0x\n",
                             zmin, zmax );
                }
                for( int j = 0; j < nitems; j++ ) {
                    item = PickBuffer[index++];
                    // item is one of your pick names
                    // do something with it >>
                    if( DEBUG )
                        fprintf( stderr, "\t%2d: %6d\n", j, item );
                }
             }
             if( Nhits == 0 ) {
                 // didn’t pick anything
                 // use the left mouse for rotation or scaling:
             }
     }
```

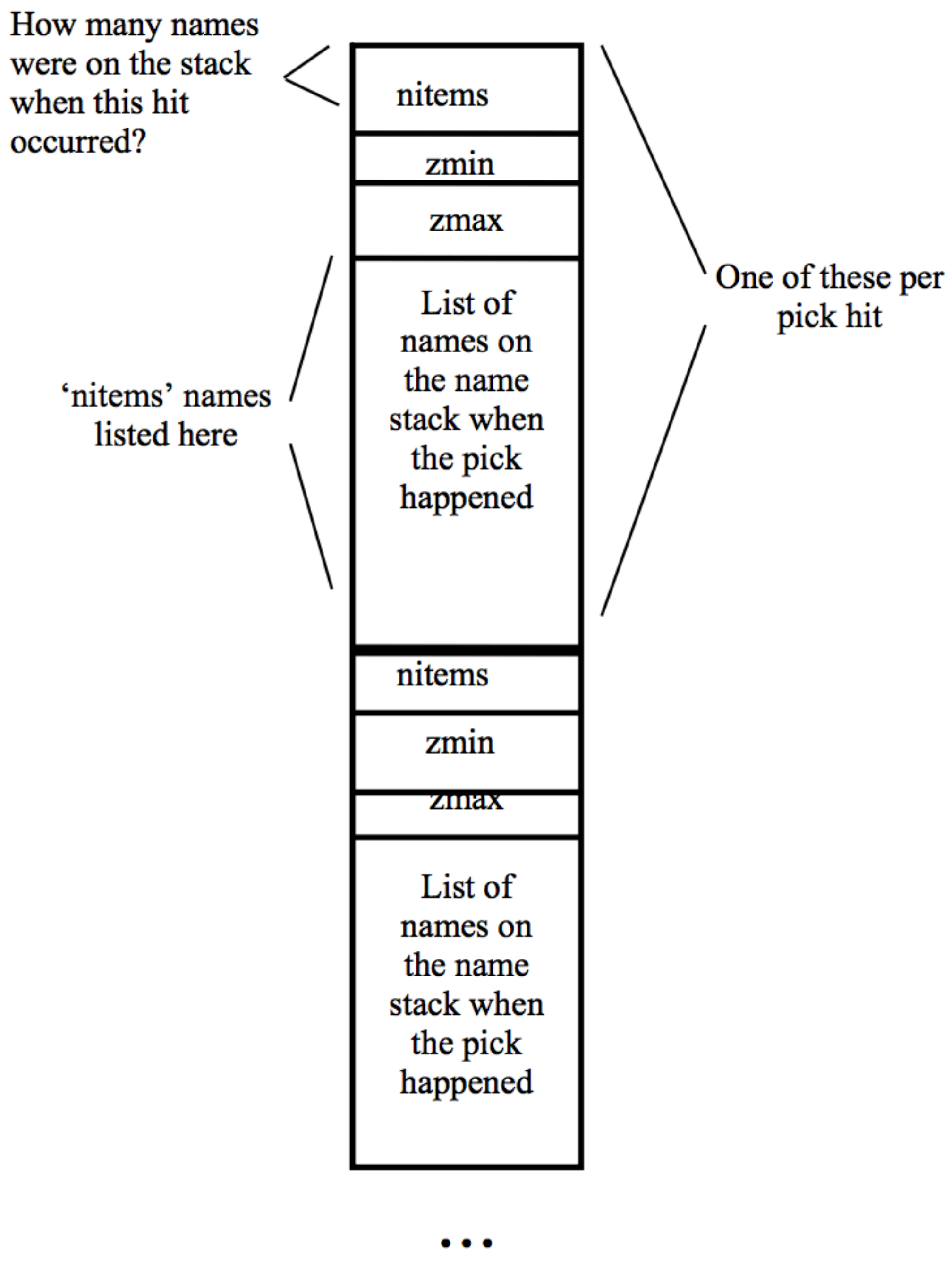
A Kludge for a Hardware Bug

Some of the TNT2-based graphics cards have a picking bug. You need to pick twice to get a pick to register correctly. You can either click everything twice (pretty lame thing to have to do), or you can add these lines into the MouseButton() code, as shown above::

```
if( Nhits == 0 ) {
    RenderMode = GL_SELECT;
    glRenderMode( GL_SELECT );
    renderScene();
    RenderMode = GL_RENDER;
    Nhits = glRenderMode( GL_RENDER );
}
```

What's In The Picking Array After a Pick Has Happened?

The elements of the picking array are organized as:


How many names were on the stack when this hit occurred?
nitems
zmin
zmax
List of names on the name stack when the pick happened
'nitems' names listed here
One of these per pick hit
nitems
zmin
zmax
List of names on the name stack when the pick happened
• • •

The zmin and zmax values are in an *internal unsigned* integer representation and are used to tell which of the objects picked is closest to you. A low value of zmin or zmax is closer to you than a high value.

renderScene()

Now, all we have to do it get `renderScene()` to do the right things. There are only a couple of steps that change depending on whether we are picking or rendering:

```
GLint viewport[4]; /* place to retrieve the viewport numbers */

glMatrixMode( GL_PROJECTION );
glLoadIdentity();
if( RenderMode == GL_SELECT ) {
      glGetIntegerv( GL_VIEWPORT, viewport );
      glm::vec4 viewportVec( viewport[0], viewport[1],
                             viewport[2], viewport[3] );
      glm::mat4 pickMtx =
          glm::pickMatrix( glm::vec2( mouseX, mouseY ),
                           glm::vec2( PICK_TOL, PICK_TOL ),
                           viewportVec );
      glMultMatrixf( &pickMtx[0][0] );
}
// call glm::ortho(), glm::frustum(), or glm::perspective()

// init the picking buffer:
if( RenderMode == GL_SELECT ) {
   glInitNames();
   glPushName( 0xffffffff );
}

// draw the objects:
  // your graphics drawing and pick name calls go here


// swap double-buffered framebuffers (in draw loop)
if( RenderMode == GL_RENDER )
    glfwSwapBuffers();
```

The only really tricky thing is this part:

```
            glGetIntegerv( GL_VIEWPORT, viewport );
            glm::vec4 viewportVec( viewport[0], viewport[1],
                                   viewport[2], viewport[3] );
            glm::mat4 pickMtx =
                glm::pickMatrix( glm::vec2( mouseX, mouseY ),
                                 glm::vec2( PICK_TOL, PICK_TOL ),
                                 viewportVec );
            glMultMatrixf( &pickMtx[0][0] );
```

This code looks at the location and size of your viewport, where the mouse is, and what picking tolerance you wanted and changes your projection matrix so that this pick box region occupies the full window. Thus, the hardware is going to check to see if *anything* gets through the clipping test and thus gets drawn. If it does, then a pick occurred.

Note that the `glm::pickMatrix()` line is written *first* in the program's transformations because we want this transformation to happen *last* after all other transformations are done.

Picking Tricks

1. Do not waste picking time looking at things you didn't want to pick anyway. For example, if the axes are not meant to *ever* be picked, do something like this:

```
     if( AxesOnOff == ON && RenderMode == GL_RENDER )
         glCallList( AxesList );
```

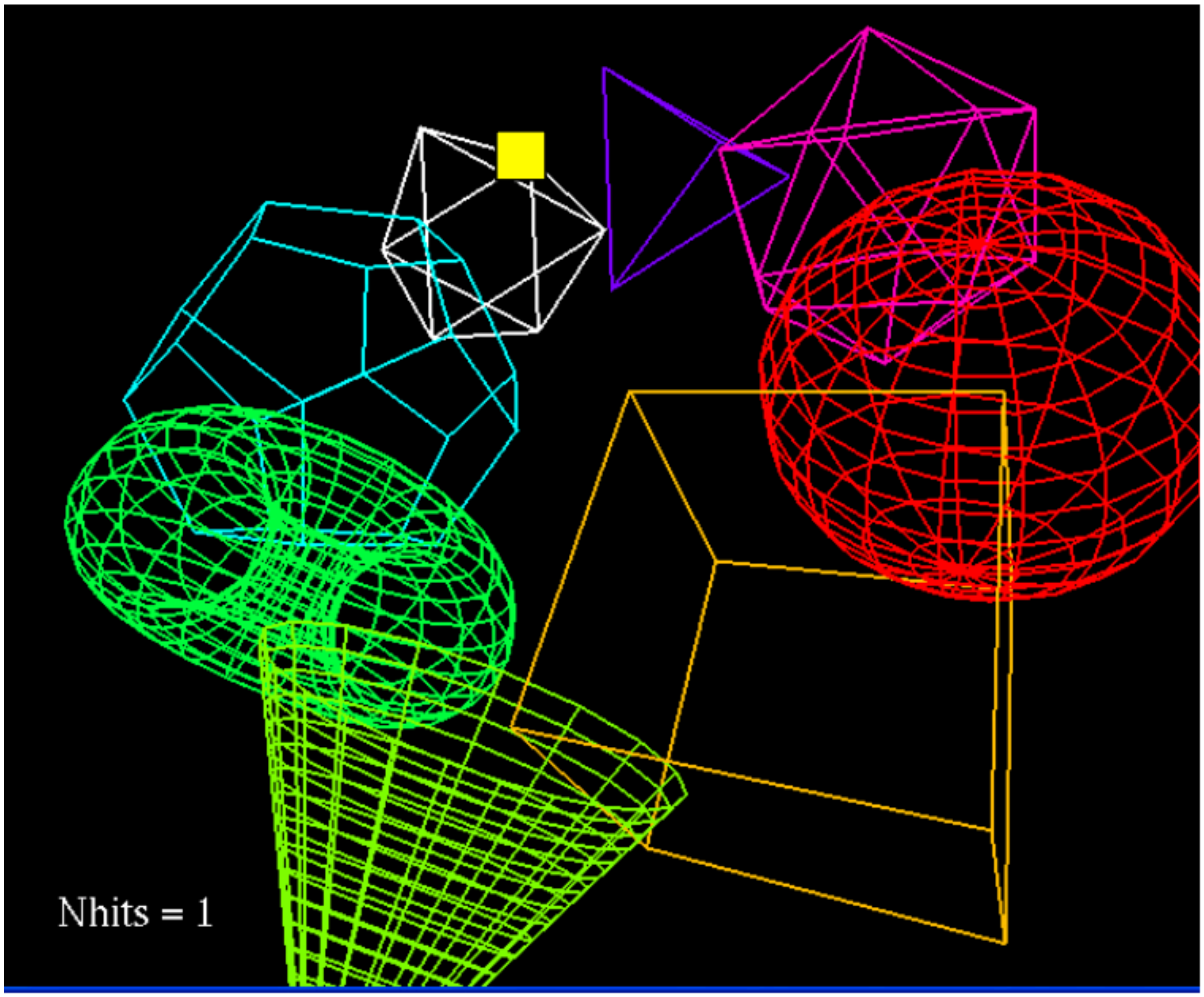
2. Recognizing that the user will never see what is being drawn during a Pick Display, you can draw something a different than what the user *thinks* he/she is seeing.

For example, a wireframe object normally cannot be picked in the empty space between the lines. But, if you draw a solid version of the same thing during the Pick Display, then it will look like a user's pick in the empty space will get the wireframe object, even though *you* know it shouldn't. So, in `renderScene()` you could say:

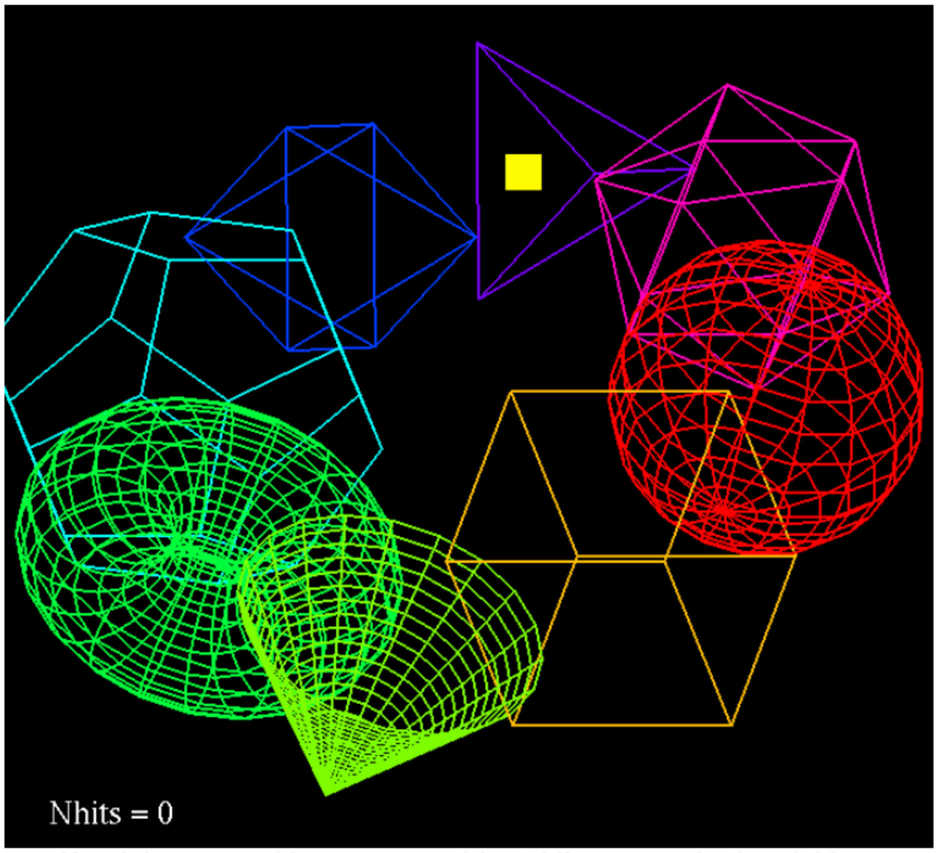
```
if( RenderMode == GL_SELECT )
      glCallList( SolidTorusList );
else
     glCallList( WireTorusList );
```

Examples

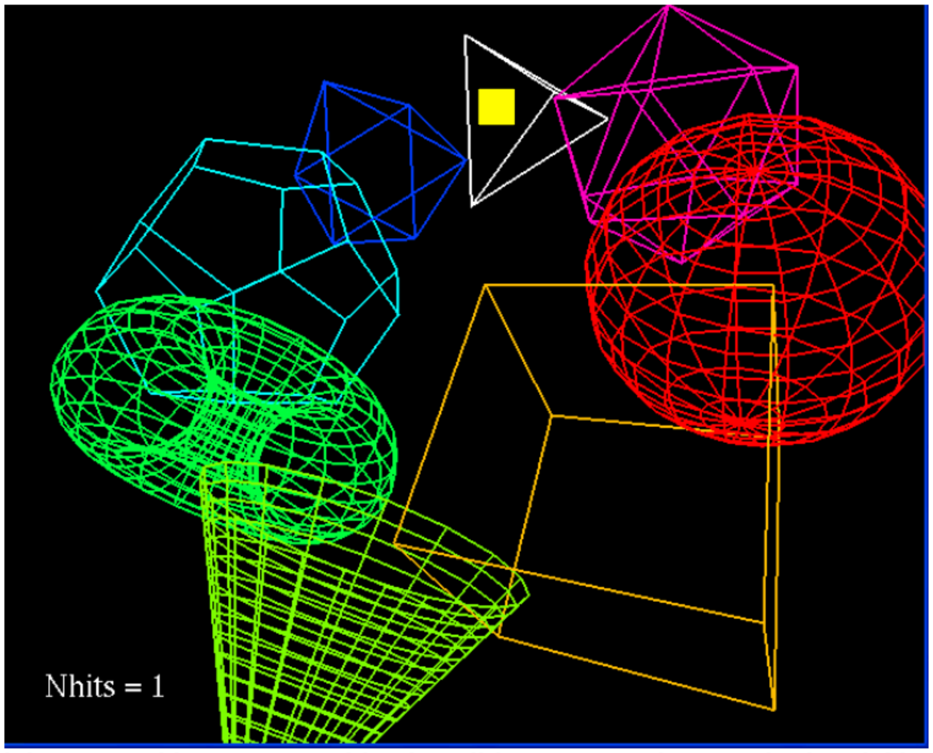
(Yellow Square = where the cursor-surrounding pickbox was when the mouse button was clicked.)



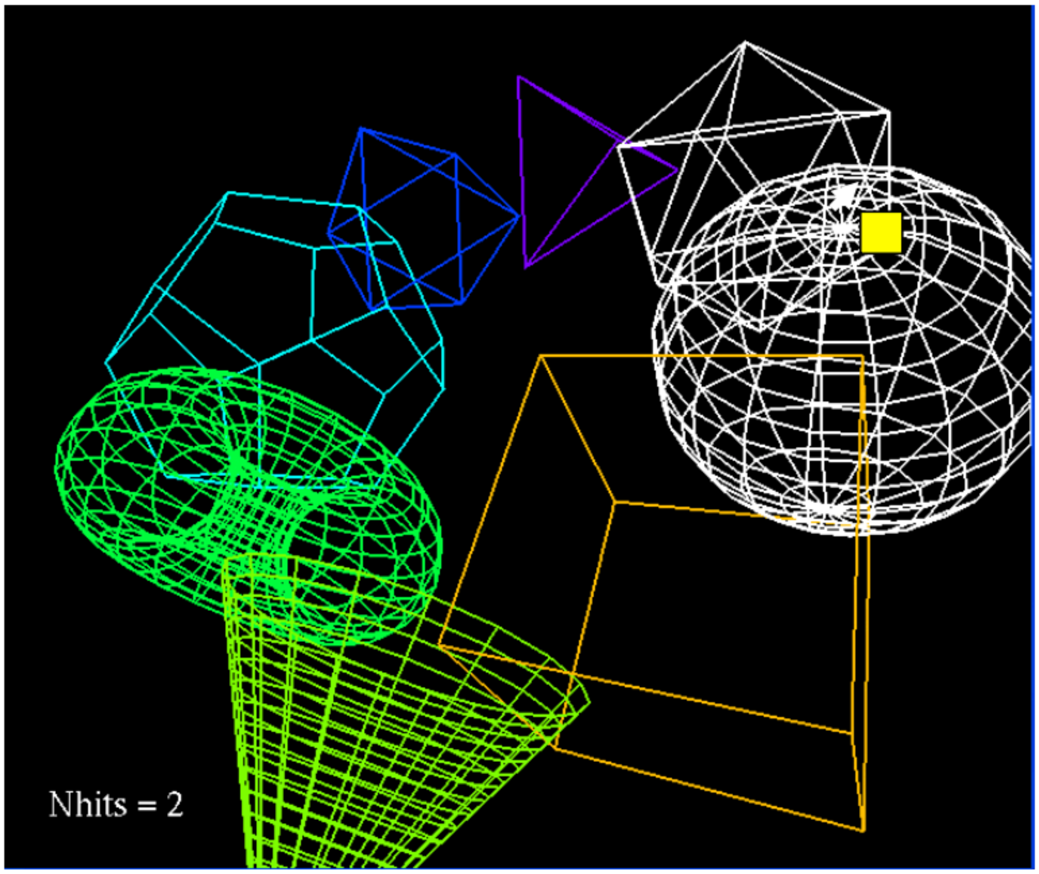
Pick occurred because an object's line was in the pickbox



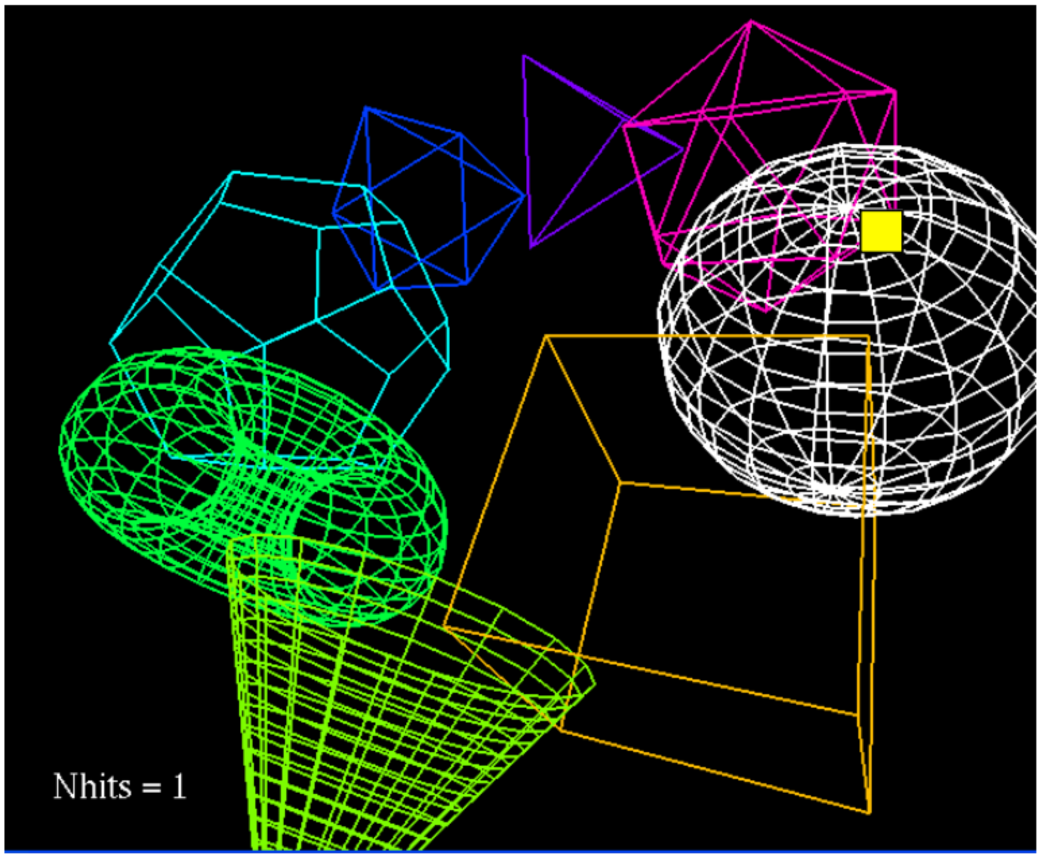
No pick occurred because no objects' lines were in the pickbox



Pick occurred because an object's solid representation was in the pickbox



Two picks occurred because two object's lines were in the pickbox



One pick occurred because it was selecting the closest object