Interaction with OpenGL — selection and picking

Selection

Selection means determining which graphical objects are within some specific volume of space. Later we shall see picking where the volume of space is given interactively by the user (usually by a mouse click). In selection, information about what is drawn is returned to the application rather than being placed in the framebuffer.

In selection, the scene is first drawn into the framebuffer, then selection mode is entered and the scene is redrawn. While in selection mode, the contents of the framebuffer remain unchanged. When selection mode exits, a list of the objects that intersect the viewing volume (typically a different volume) is returned.

The list returned—the hit record—is an array of integer identifiers corresponding to entries in the name stack.

The name stack is constructed in selection mode by loading names (integers) as drawing commands are issued.

The returned list indicates which objects might have been selected on the screen by the user.

To use the selection mechanism, the following steps are performed:

1. The array to be used for the returned hit records is specified with `glSelectBuffer()`.
2. Selection mode is entered by specifying `GL_SELECT` with `glRenderMode()`.
3. The name stack is initialized using `glInitNames()` and `glPushName()`.
4. The viewing volume to use for selection is defined. (Since this is generally different from the viewing volume originally used to draw the scene, the current transformation state should be saved with `glPushMatrix()` and restored with `glPopMatrix()`.)
5. Drawing commands and commands to manipulate the name stack are issued so that each primitive of interest has an appropriate name assigned.
6. Exit selection mode and process the returned selection data (the hit records).

`void glSelectBuffer(GLsizei size, GLuint *buffer);`

Specifies the array to be used for the returned selection data; `buffer` is a pointer to an array of unsigned integers into which the data is put, and `size` indicates the maximum number of values that can be stored in the array. `glSelectBuffer()` must be called before entering selection mode.

`GLint glRenderMode(GLenum mode);`

Controls whether the application is in rendering, selection, or feedback mode (not covered). The mode argument can be one of `GL_RENDER` (the default), `GLSELECT`, or `GL_FEEDBACK`. The program remains in a given mode until `glRenderMode()` is called again with a different argument. Before entering selection mode, `glSelectBuffer()` must be called to specify the selection array. The return value is the number of selection hits or the number of values placed in the feedback array when either mode is exited; a negative value means that the selection or feedback array has overflowed. `GLRENDER_MODE` can be used with `glGetlntegerv()` to obtain the current mode.

Creating the Name Stack

The name stack is the basis for the selection information that is returned. It is first initialized with `glInitNames()`, which simply clears the stack, and then integer names are added to it while issuing corresponding drawing commands.

`glPushName()`, and `glPopName()`, allow pushing and popping of the stack; `glLoadName()` replaces the name on the top of stack with its argument.

The following code shows simple name-stack manipulation:

```
glInitNames ();
glPushName(0);
glPushMatrix();   /* save the transformation matrix */

    /* create your desired viewing volume here */

glLoadName(1);
drawFirstObject();   /* FirstObject has the name 1 */
glLoadName(2);
drawSecondObject();  /* SecondObject and ThirdObject */
drawThirdObject();   /* both have the same name - 2 */

glPopMatrix ();   /* restore previous transformation */
```

Notes:

- The call `glPushName(0)` loads the 'dummy' name, 0. Without some existing name on the stack the first call to `glLoadName` would fail.
- Multiple objects can share the same name if there is no need to distinguish between them.

`void glPushName(GLuint name)`

Pushes `name` onto the name stack. Pushing beyond the stack depth generates the `GL_STACK_OVERFLOW` error. `GL_NAME_STACK_DEPTH` can be used with `glGetIntegerv()` to obtain the depth of the name stack. The depth of the name stack is at least sixty-four names.

`void glPopName(void)`

Pops one name off the top of the name stack. Popping an empty stack generates the error `GL_STACK_UNDERFLOW`.

`void glLoadName(GLuint name)`;

Replaces the value on the top of the name stack with `name`. If the stack is empty, (e.g., right after `glInitNames()` is called) `glLoadName()` generates the error `GL_INVALID_OPERATION`. `glPushName()` should be called at least once to put something on the name stack before calling `glLoadName()`.

Calls to `glPushName()`, `glPopName()`, and `glLoadName()` are ignored when not in selection mode. This can simplify coding by using these calls in the drawing code, and then using the same drawing code in both selection and normal rendering modes.

The Hit Record

In selection mode, a primitive that intersects the viewing volume causes a selection hit. Only one hit will be written into the hit record between calls to manipulate the name stack, or to `glRenderMode`. Thus, objects sharing the same name will not generate multiple hit records. Hit records are not guaranteed to be written into the array until `glRenderMode()` is called.

Each hit record consists of four items, in order:

- The number of names on the name stack when the hit occurred.
- Both the minimum and maximum window-coordinate z values of all vertices of the primitives that intersected the viewing volume since the last recorded hit. These two values, lying in the range $[0, 1]$, are each multiplied by $2^{32} - 1$ and rounded to the nearest unsigned integer.
- The contents of the name stack at the time of the hit, with the bottommost element first.

Upon entering selection mode, OpenGL initializes a pointer to the beginning of the selection array. Each time a hit record is written into the array, the pointer is updated accordingly. When selection mode is exited with `glRenderMode()`, the name stack is cleared and the number of hit records is returned. If the hit record overflows then -1 is returned.

Note that the z values in the hit record are represented as integers. Dividing these by the largest unsigned integer gives the actual depth in the window. To get the depth in object coordinates, the transformation matrix parameters (e.g., the arguments to `glOrtho()`, or `gluPerspective()`) are required.

A Selection Example

In the sample program `select.c`, four triangles (green, red, and two yellow triangles, are created by calling `drawTriangle()`) and a wireframe box showing the viewing volume (`drawViewVolume()`) are drawn to the screen. Then the triangles are rendered again (`selectObjects()`), in selection mode. The corresponding hit records are processed in `processHits()`, where the hit list is printed out. The first triangle generates a hit, the second does not, and the third and fourth together generate a single hit.

Look at this code carefully to understand its operation.

```
/* draw a triangle with vertices at (x1, y1), (x2, y2)
 * and (x3, y3) at z units away from the origin.  */
void drawTriangle (GLfloat x1, GLfloat y1,
                   GLfloat x2, GLfloat y2,
                   GLfloat x3, GLfloat y3, GLfloat z) {
   glBegin (GL_TRIANGLES);
   glVertex3f (x1, y1, z);
   glVertex3f (x2, y2, z);
   glVertex3f (x3, y3, z);
   glEnd ();
}
```

```
void draw4Triangles (void)
{
   glLoadName(1);   /* ignored in render mode */

   glColor3f (0.0, 1.0, 0.0);   /*  green triangle   */
   drawTriangle (2.0, 2.0, 3.0, 2.0, 2.5, 3.0, -5.0);

   glLoadName(2);

   glColor3f (1.0, 0.0, 0.0);   /*  red triangle     */
   drawTriangle (2.0, 7.0, 3.0, 7.0, 2.5, 8.0, -5.0);

   glLoadName(3);

   glColor3f (1.0, 1.0, 0.0);   /*  yellow triangles */
   drawTriangle (2.0, 2.0, 3.0, 2.0, 2.5, 3.0, 0.0);
   drawTriangle (2.0, 2.0, 3.0, 2.0, 2.5, 3.0, -10.0);
}
```

```
void drawScene (void)
{
   glMatrixMode (GL_PROJECTION);
   glLoadIdentity ();
   gluPerspective (40.0, 4.0/3.0, 1.0, 100.0);

   glMatrixMode (GL_MODELVIEW);
   glLoadIdentity ();
   gluLookAt (7.5, 7.5, 12.5, 2.5, 2.5, -5.0,
                0.0, 1.0, 0.0);

   draw4Triangles();

   drawViewVolume (0.0, 5.0, 0.0, 5.0, 0.0, 10.0);
  }
```

`drawViewVolume()` draws a view volume which will be used as the clipping volume in the next function, `selectObjects()`.

```
void selectObjects(void) {
   GLuint selectBuf[BUFSIZE];
   GLint hits;

   glSelectBuffer (BUFSIZE, selectBuf);
   glRenderMode (GL_SELECT); /* set SELECT mode */
   glInitNames();    /* initialize name stack  */
   glPushName(0);
   glPushMatrix ();
   glMatrixMode (GL_PROJECTION);
   glLoadIdentity ();
   glOrtho (0.0, 5.0, 0.0, 5.0, 0.0, 10.0);
   glMatrixMode (GL_MODELVIEW);
   glLoadIdentity ();

   draw4Triangles();

   glPopMatrix ();
   glFlush ();
   hits = glRenderMode (GL_RENDER); /*  no. of hits */
   processHits (hits, selectBuf);
  }
```

```
void processHits (GLint hits, GLuint buffer[])
{
  unsigned int i, j;
  GLuint names, *ptr;
  float z1, z2;

  printf ("hits = %d\n", hits);
  ptr = (GLuint *) buffer;
  for (i = 0; i < hits; i++) { /*  for each hit  */
    /* ptr points to hit list - returns no. of names,
       min and max z coordinates, and each name in the
       name stack, bottommost first  */
    names = *ptr; ptr++;
    z1 = (float) *ptr/0xffffffff; ptr++;
    z2 = (float) *ptr/0xffffffff; ptr++;
    printf(" number of names for hit = %d\n", names);
    printf("  z1 is %g;  z2 is %g\n", z1, z2);
    printf ("   the name is ");
    for (j = 0; j < names; j++)     /*  for each name */
      printf ("%d ", *ptr); ptr++;
    printf ("\n");
  }
}
```

Note that the buffer containing the hit record was defined with

```
glSelectBuffer(BUFSIZE, selectBuf);
```

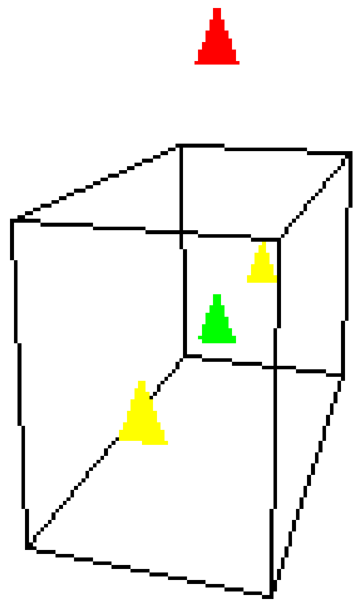
and contains data of type `GLuint` relating to the hit.

The code shown is "boiler plate" C code for obtaining values from the hit record.

The output is basically the contents of the hit record, in the order in which it was written. Note that in this example, there is only one name for each item.

Usually, the information in the hit record would be used to trigger some other event; e.g., highlighting the items in the view volume.

Sample output for `select.c`



```
hits = 2
 number of names for hit = 1
  z1 is 0.5;  z2 is 0.5
   the name is 1
 number of names for hit = 1
  z1 is 0; z2 is 0.999999
   the name is 3
```

Recall that the green triangle had label 1, the red triangle had label 2 and the two yellow triangles had label 3.

Clearly, the select operation found the green and yellow triangles, but did not select the red triangle.

Picking

The picking operation is really just a special case of selection. A special picking matrix is used in conjunction with the projection matrix to restrict drawing to a small region of the viewport, typically near the cursor.

Some form of input, such as clicking a mouse button, initiates selection (pick) mode. With selection mode set, together with the special picking matrix, objects drawn near the cursor cause selection hits. Thus, during picking you are typically determining which objects are drawn near the cursor.

Picking is set up similarly to regular selection mode, with the following major differences:

- Picking is usually triggered by an input device. In the following code examples, pressing the left mouse button invokes a function that performs picking.
- The utility routine `gluPickMatrix()` is used to multiply a special picking matrix onto the current projection matrix.

`gluPickMatrix()` should be called just before setting a standard projection matrix (such as `gluPerspective()` or `glOrtho()`).

Normally, the contents of the projection matrix are first saved, so the sequence of operations may look like this:

```
glMatrixMode (GL_PROJECTION);
glPushMatrix ();
glLoadIdentity ();
gluPickMatrix (...);
gluPerspective, glOrtho, glOrtho2D, or glFrustum
    /* ... draw scene for picking  */
    /* ... perform picking ...     */
glPopMatrix();
```

`void gluPickMatrix(GLdouble x, GLdouble y, GLdouble width, GLdouble height, GLint viewport[4]);`

Creates a projection matrix that sets the selection volume to a small sub-volume of the volume specified by `gluPerspective` or `glOrtho`. This sub-volume (which has some specification in eye coordinates) and is centred on (`x`, `y`) in window coordinates. Typically, (`x`, `y`) is the cursor location. `width` and `height` define the size of the picking region in screen coordinates. `viewport[]` indicates the current viewport boundaries, obtained by calling `glGetIntegerv(GL_VIEWPORT, GLint *viewport);`

The net result of the matrix created by `gluPickMatrix()` is to transform the pick volume into the canonical clipping region $-1 < (xyz) < 1$.

In certain situations, you might need to specify additional clipping planes to further refine the picking region.

The program `pickdepth.c` illustrates simple picking of overlapping elements.

In this example, when the left mouse button is pressed, the function `pickRects()` is called to identify which rectangles were picked by the mouse.

Note that for one press of the mouse key, several hits (corresponding to the different rectangles under the cursor) may be returned.

This code should be examined carefully.

Sample code from pickdepth.c (slightly simplified)

```
void myinit(void)
{
  glClearColor(0.0, 0.0, 0.0, 0.0);
  glEnable(GL_DEPTH_TEST);
  glShadeModel(GL_FLAT);
}



void display(void)
{
  glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);
  drawRects();
  glutSwapBuffers();
}
```

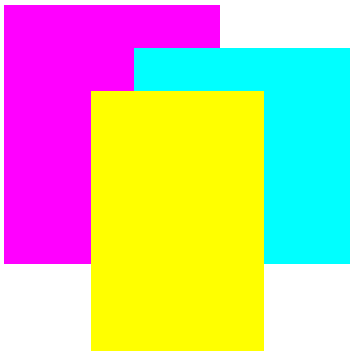
The function **`drawRects()`** draws 3 overlapping rectangles, with colors yellow, cyan, and magenta named 1, 2, and 3, respectively.

```
void drawRects()
/* slightly simplified from the original */
{
    glLoadName(1);
  glBegin(GL_QUADS);
  glColor3f(1.0, 1.0, 0.0);  /* yellow */
  glVertex3i(2, 0, 0); glVertex3i(2, 6, 0);
  glVertex3i(6, 6, 0); glVertex3i(6, 0, 0);
  glEnd();
    glLoadName(2);
  glBegin(GL_QUADS);
  glColor3f(0.0, 1.0, 1.0);  /* cyan */
  glVertex3i(3, 2, -1); glVertex3i(3, 7, -1);
  glVertex3i(8, 7, -1); glVertex3i(8, 2, -1);
  glEnd();
    glLoadName(3);
  glBegin(GL_QUADS);
  glColor3f(1.0, 0.0, 1.0);  /* magenta */
  glVertex3i(0, 2, -2); glVertex3i(0, 8, -2);
  glVertex3i(5, 8, -2); glVertex3i(5, 2, -2);
  glEnd();
}
```

```
void pickRects(int button, int state, int x, int y) {
  GLuint selectBuf[BUFSIZE];
  GLint hits;
  GLint viewport[4];
  if (button != GLUT_LEFT_BUTTON || state != GLUT_DOWN)
    return;

  glGetIntegerv(GL_VIEWPORT, viewport);
  glSelectBuffer(BUFSIZE, selectBuf);
  glRenderMode(GL_SELECT);
  glInitNames();
  glPushName(-1);

  glMatrixMode(GL_PROJECTION);
  glPushMatrix();
  glLoadIdentity();
  gluPickMatrix((GLdouble) x, (GLdouble) (viewport[3]-y),
    5.0, 5.0, viewport);
  glOrtho(0.0, 8.0, 0.0, 8.0, -0.5, 2.5);
  drawRects();
  glPopMatrix();
  glFlush();
  hits = glRenderMode(GL_RENDER);
  processHits(hits, selectBuf);
}
```

```
void processHits(GLint hits, GLuint buffer[])
/* this is the same "boiler plate" as before */
{
  unsigned int i, j;
  GLuint names, *ptr;
  float z1, z2;

  printf("hits = %d\n", hits);
  ptr = (GLuint *) buffer;
  for (i = 0; i < hits; i++) {  /* for each hit  */
    names = *ptr;  ptr++;
    z1 = (float) *ptr/0xffffffff; ptr++; /* scale 0-1 */
    z2 = (float) *ptr/0xffffffff; ptr++;
    printf(" number of names for hit = %d\n", names);
    printf("  z1 is %g;  z2 is %g\n", z1, z2);
    printf("   the name is ");
    for (j = 0; j < names; j++) {  /* for each name */
      printf("%d ", *ptr); ptr++;
    }
    printf("\n");
  }
}
```

sample output from `pickdepth.c`



For a cursor click in the center:

```
hits = 3
 number of names for hit = 1
  z1 is 0.166667;  z2 is 0.166667
   the name is 1
 number of names for hit = 1
  z1 is 0.5;  z2 is 0.5
   the name is 2
 number of names for hit = 1
  z1 is 0.833333;  z2 is 0.833333
   the name is 3
```

Using Multiple Names in a Hierarchical Model

Multiple names can also be used to distinguish between parts of a hierarchical object in a scene.

For example, if you were rendering an assembly line of automobiles, you might want the ability to move the mouse to pick the third bolt on the left front tire of the third car in line. Since the bolts are identical, it is simplest to have the same name for each bolt, and to establish the particular bolt picked using a name hierarchy. A different name can be used to identify each level of hierarchy: which car, which tire, and finally which bolt.

As another example, one name could be used to describe a single molecule among other molecules, and additional names can differentiate individual atoms within that molecule.

The following code snippet draws an automobile with four identical wheels, each of which has five identical bolts. Code has been added to manipulate the name stack with the object hierarchy.

```
/* Example showing the creation of multiple names  */
void draw_wheel_and_bolts()
{
   long i;

   draw_wheel_body();
   for (i = 0; i < 5; i++) {
      glPushMatrix();
         glRotate(72.0*i, 0.0, 0.0, 1.0);
         glTranslatef(3.0, 0.0, 0.0);
         glPushName(i);
            draw_bolt_body();
         glPopName ();
      glPopMatrix();
   }
}
```

```
draw_body_and_wheel_and_bolts()
{
  drawcarbody();
  glPushMatrix();
     glTranslate(40, 0, 20);  /* first wheel position */
     glPushName(1);       /* name of wheel number 1 */
        draw_wheel_and_bolts();
     glPopName ();
  glPopMatrix();

  glPushMatrix();
     glTranslate(40, 0, 20); /* second wheel position */
     glPushName(2);          /* name of wheel number 2 */
        draw_wheel_and_bolts();
     glPopName C);
  glPopMatrix ();

/* draw last two wheels similarly */
}
```

The following example uses the functions defined above to draw three different cars, numbered 1, 2, and 3:

```
draw_three_cars ()
{   glInitNames ();
    glPushMatrix();
        translate_to_first_car_position();
        glPushName(1);
            draw_body_and_wheel_and_bolts ();
        glPopName();
    glPopMatrix();

    glPushMatrix();
        translate_to_secondcar_position ();
        glPushName(2);
            draw_body_and_wheel_and_bolts);
        glPopName ();
    glPopMatrix();

    glPushMatrix ()
        translate_to_third_car_position();
        glPushName(3);
            draw_body_and_wheel_and_bolts();
        glPopName ();
    glPopMatrix();
}
```

Assuming that picking is performed, the following are some possible name-stack return values and their interpretations. Each entry in the table below gives the number of names per hit; the min. and max. z-values (`d1` and `d2`); and then the contents of the name stack.

`2 d1 d2 2 1`	Car 2, wheel 1
`1 d1 d2 3`	Car 3 body
`3 d1 d2 1 1 0`	Bolt 0 on wheel 1 on car I
`empty`	The pick was outside all cars