Picking and Highlighting

Interactive selection of objects with feedback is an important part of modeling applications. OpenGL provides several mechanisms which can be used to do the perform the object selection and the highlighting tasks.

1 OpenGL Selection

OpenGL supports an object selection mechanism in which the object geometry is transformed and compared against a selection subregion of a window. The mechanism uses the transformation pipeline to compare object vertices against the view volume. To reduce the view volume to a screen-space subregion (window coordinates), the projected coordinates of the object are transformed by the following matrix

\[
T = \begin{pmatrix}
\frac{p_x}{d_x} & 0 & 0 & p_x - \frac{d_x}{2} \\
0 & \frac{p_y}{d_y} & 0 & p_y - \frac{d_y}{2} \\
0 & 0 & 1 & 0 \\
0 & 0 & 0 & 0
\end{pmatrix}
\]

where \(o_x, o_y, p_x,\) and \(p_y\) are the \(x\) and \(y\) origin and width and height of the viewport, and \(q_x, q_y, d_x,\) and \(d_y\) are the origin and width and height of the pick region.

Objects are identified by assigning them integer names using `glLoadName()`. As each object is sent to the OpenGL pipeline and tested against the pick region, if the test succeeds a hit record is created to identify the object. The hit record is written to the selection buffer whenever a change is made to the current object name. An application can determine which objects intersected the pick region by scanning the selection buffer and examining the names present in the buffer.

The OpenGL selection method determines that an object has been hit if it intersects the view volume. Bitmap and pixel image primitives will generate a hit record only if a raster positioning command is sent to the pipeline and the transformed position lies within the viewing volume. To generate hit records for any point within a pixel image or bitmap, a bounding rectangle should be sent rather than the image, and selection testing performed on the interior of the rectangle. Similarly, wide lines and points will be selected only if the equivalent infinitely thin line or small point would be selected. To facilitate selection testing of wide lines and points, proxy geometry representing the true footprint of the primitive should be used instead.

Many applications take advantage of instancing of geometric data to reduce the memory footprint. Instancing allows an application to create a single representation of the geometric data for each type of object used in the scene. If the application is modeling a car, the four wheels of the car may be represented as instances of a single geometric description of a wheel combined with a modeling transformation to place the wheel in the correct location in the scene. Instancing introduces extra complexity into the picking operation. If a single name is associated with the wheel geometry, the application can not determine which of the four instances of the wheel has been picked. OpenGL solves this problem by maintaining a stack of object names. This allows an application that represents a model hierarchically to associate a name at each stage in the hierarchy. As the hierarchical model is drawn new names are pushed onto the stack as the hierarchy is descended and old names are popped as the hierarchy is ascended. When a hit record is created, it contains all of
the names currently in the name stack. The application can determine which instance of an object is selected by looking at the contents of the name stack and comparing them to the names stored in the hierarchical representation of the model.

Using the car model example, the application associates an object name with the wheel representation and another object name with each of the transformations used to position the wheel in the car model. The application determines that a wheel is selected if the selection buffer contains the object name for the wheel, and it determines which instance of the wheel by examining the object name of the transformation.

When the OpenGL pipeline is in selection mode, the primitives sent to the pipeline do not generate fragments to the framebuffer. Since only the result of the transformation pipeline is of interest, there is no need to send texture coordinates, normals, etc. or to enable lighting.

2 Object Tagging in the Color Buffer

An alternative method for locating objects is to write integer object names as color values into the framebuffer and read back the framebuffer data within the pick region to reconstruct the object names. In order for this to work correctly, the application needs to be able to predictably write and read color values. Texturing, blending, dithering, lighting and smooth shading should be disabled so that fragment color values are not altered during rasterization or fragment processing. The unsigned integer forms of the color commands (glColor3ub(), etc) should be used to pass in the object names, as the unsigned forms are specified to convert the values in such a way as to preserve the $b$ most significant bits of the color value, where $b$ is the number of bits in the color buffer. To limit selection to visible surfaces, depth testing should be enabled. The back color buffer can be used for the drawing operations to keep the drawing operations invisible to the user.

A typical RGB color buffer storing 8-bit components can represent 24-bit object names. However, since the color must uniquely identify the object, instancing information must be available directly in the color. That is, the functionality provided by the name stack in the OpenGL selection mechanism is not available, so the application may need to partition the representable name space to hold hierarchy information. For example a 4 level hierarchy can be allocated a 24-bit color as 4,4,6 and 10 bits allowing more object names for leaf parts of the hierarchy.

One disadvantage of using the color buffer is that the color buffer can only hold a single identifier at each pixel. If depth buffering is used, then the pixel will hold the object name corresponding to a visible surface. If depth buffering is not used, then a pixel hold the name of the last surface drawn. The OpenGL selection mechanism can return a hit record for all objects that intersect a given region. The application is free to choose one of the intersecting objects using a separate policy, e.g. the object closest to the viewer, iterate through all of the objects one at a time, etc.

3 Proxy Geometry

One method to reduce the amount of work done by the OpenGL pipeline during picking operations (for color buffer tagging or OpenGL selection) is to use a simplified form of the object in the picking computations. For example, individual objects can be replaced by geometry representing their bounding boxes. The accuracy of the picking operation is traded for increased speed. Some of the accuracy can be improved by adding a second pass in which the objects which are selected using their simplified geometry are reprocessed using their real geometry.

4 Other Methods
For many applications it may prove advantageous to not use the OpenGL pipeline at all to implement picking. For example, an application may choose to organize its geometric data spatially and use a hierarchy of bounding volumes to efficiently prune portions of the scene without testing each individual object.

5 Highlighting

Once the selected object has been identified, an application will typically modify the appearance of the object to indicate that it has been selected. A simple way to accomplish this is to redraw the entire scene, drawing the selected object with a different appearance (wireframe, different color, etc).

In applications manipulating complex models, the cost of redrawing the entire scene to indicate a selection may be prohibitive. This is particularly true for applications which implement locate-highlight in which each object is highlighted as the cursor passes over or near it to indicate that this is the candidate object for manipulation by the application. An extension of this problem exists for painting applications that need to track the location of a brush over an object and make changes to the appearance of the object based on the current painting parameters.

An alternative to redrawing the entire scene is to use overlay color buffers to draw highlights on top of the existing scene. One difficulty with this strategy is that it may be difficult to modify only the visible surfaces of the selected object since the depth information is present in the depth buffer associated with the main color buffer. For applications in which the visible surface information is not required, overlay buffers are an efficient solution. If visible surface information is important, then it may be better to modify the color buffer directly. A selected object that has been depth buffered can be overdrawn directly by changing the depth test function to `GL_LEQUAL` and redrawing the object geometry with different attributes.

6 XOR Highlighting

Another efficient highlighting technique is to overdraw with an XOR logic operation. An advantage of using XOR is that the highlighting and restoration operations can be done independently of the original object color. The most significant bit of each of the color components can be XORed to produce a large difference between the highlight color and the original color. Drawing a second time restores the original color.

A second advantage of the XOR method is that depth testing can be disabled to allow the non-visible surfaces to poke through occluding objects. The highlight can be later removed without needing to redraw the occluders.

One should also be careful of interactions between the picking and highlighting methods. For example, a picking mechanism that uses the color or depth buffer can not be mixed with a highlighting algorithm that relies on the contents of those buffers remaining intact between highlighting operations.

A useful hybrid scheme for combining color buffer tagging with locate-highlight on visible surfaces is to share the depth buffer between the picking and highlighting operations and to use the front color buffer for highlighting operations and the back color buffer for locate operations. Each time the viewing or modeling transformations change, the scene is redrawn updating both color buffers and locate-highlight operations are performed using the same buffers until another modeling or
viewing change requires a redraw. This type of algorithm can be very effective for achieving interactive rates for complex models since very little geometry needs to be rendered between modeling and viewing changes.