OpenGL Programming Guide, Chapter 2

R. J. Renka

Department of Computer Science & Engineering
University of North Texas

09/09/2013
There are three basic drawing operations.

1. Clearing the window means initializing the contents of buffers. Computer monitors use memory-mapped graphics in which pixel color values are defined by the contents of the framebuffer.

2. Drawing raster objects: two-dimensional images, bitmaps, and character fonts. The view volume might coincide with the window in this case.

3. Drawing geometric objects created from points, lines (line segments), and polygons (usually triangles). A curve is approximated by a polygonal line, and a curved surface is approximated by a polygonal surface (usually a triangle mesh) with a lot of triangles, and maybe smooth shading.
There are two methods for storing pixel colors.

1. **Indexed**: A single index into a *color lookup table* implemented in software. This is not supported by all OpenGL implementations, and therefore should not be used.

2. **RGBA**: Red, Green, Blue, and Alpha values, where the alpha channel is used for blending (transparency) — Chapter 6. By default, the new pixel color overwrites the previous color. If blending is enabled, the usual option is to write $\alpha F + (1 - \alpha)B$, where $F$ is the foreground (source) color, $B$ is the background (destination) color, and $\alpha$ is the source alpha value. Since, for example, $\alpha = .2$ implies 80% transparent, the alpha value is actually *opacity* rather than transparency. Its default value is 1. The default background color is 0 but can be changed by `glClearColor`.
Clearing the Window

The screen or window must be cleared to some background color before drawing begins except in the unlikely event that the scene covers every pixel, or in the case of XOR-mode animation.

If the view volume coincides with the entire window, we could clear the window by drawing a rectangle that covers the view volume, but it might leave gaps, and it would be less efficient than a special-purpose clearing command. Also, additional efficiency may be gained by clearing more than one buffer with a single command. Clearing buffers is expensive.

OpenGL provides front and back color buffers for double buffering, possibly left and right color buffers for stereo, a depth buffer for hidden surface removal, a stencil buffer used to restrict drawing to a specified portion of the screen, and an accumulation buffer used to accumulate a series of images. See Chapter 10.
The following command clears both the color buffers and the depth buffer.

```
glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);
```

The default depth is 1, but that can be changed by `glClearDepth`. Calls to `glClearColor` and `glClearDepth` should be placed in an initialization function that is called once before entering the event loop. A call to `glClear` should be placed at the beginning of the display callback.
Chapter 4 discusses color.

Chapter 5 discusses lighting.

Current color and lighting parameters are defined by state variables. The current color is used for all drawing unless lighting is enabled, in which case the current color is irrelevant.

`glColor*()` takes 3 or 4 arguments (any of the 8 types): R, G, B, A with A = 1 by default.
Forcing Completion of Drawing

When the CPU issues a drawing command, it enters the graphics pipeline which consists of many concurrently executing hardware units — transformations, clipping, lighting and shading, texturing, etc. The CPU does not wait for a command to finish executing before issuing a new command.

Also, due to network transmission overhead, it would be inefficient to send commands over a network one at a time. Commands are therefore accumulated into network packets for transmission.

glFlush() forces the client to send the network packet even if it is not full, and thus force drawing to begin, but does not wait for it to finish. An alternative is glFinish() which waits for all previous commands to finish, allowing synchronization of tasks; e.g., finish drawing before overwriting with labels, or copying to a file, or getting user input. If double buffering is used for smooth animation, glutSwapBuffers should be used in place of glFlush.
Defn A *view volume* is a rectangular box or frustum-shaped region in $\mathbb{R}^3$ against which all geometric primitives will be clipped.

Defn A *viewport* is a rectangular portion of the window into which the projected view volume is mapped. Nothing is drawn outside the viewport.

The reshape callback function should define both the view volume and the viewport. The default view volume is $[-1,1]^3$, and the default viewport is the entire window.

```c
void glOrtho(GLdouble l, GLdouble r, GLdouble b, GLdouble t, GLdouble n, GLdouble f);
```
defines the view volume to be $[l, r] \times [b, t] \times [-f, -n]$.

```c
void glViewport(GLint x, GLint y, GLsizei width, GLsizei height);
```
creates a viewport with lower left corner at $(x,y)$. 

R. J. Renka
OpenGL Programming Guide, Chapter 2
Two-dimensional screen-oriented applications like paint programs, image processing, and anything needing exact rasterization should have the view volume coincide with the viewport so that pixel centers are half-way between the integers. The following code sets both the view volume and viewport to the entire $w$ by $h$ window so that both eye coordinates and window coordinates are in $[0, w] \times [0, h]$ with lower left pixel centered at $(.5,.5)$, upper right at $(w-.5,h-.5)$.

```c
void reshape(int w, int h)
{
    glViewport(0, 0, (GLsizei)w, (GLsizei)h);
    glMatrixMode(GL_PROJECTION);
    glLoadIdentity();
    glLoadIdentity();
    gluOrtho2D(0.0, (GLdouble)w, 0.0, (GLdouble)h);
}
```
Real numbers are approximated by *floating-point numbers* either in 4-byte single precision (7 decimal digits) or 8-byte double precision (16 digits).

**Vertices** are represented by 2, 3, or 4 floating-point coordinates \((x,y,z,w)\) with defaults \(z = 0\) and \(w = 1\): `glVertex*` must be called within a `glBegin()`, `glEnd()` pair.

**Points** are represented by vertices and drawn with one or more pixels.

**Lines** are represented by pairs of vertices (line segments) or sequences of vertices (polygonal lines), and are rendered with some thickness and possibly antialiasing.

The OpenGL geometric drawing primitives are defined in Tables 2-2 and 2-3.
Polygons are represented by sequences of vertices defining simple (non-intersecting) closed polygonal curves in a plane such that the region enclosed (the polygon) is convex — for every pair of points in the region, the associated line segment is in the region.

Note that, for efficiency, OpenGL does not test that a sequence of vertices defines a valid polygon. A quadrilateral with non-coplanar vertices, for some eye positions and orientations, will project to a nonconvex polygon and be rendered incorrectly. Curved surfaces should therefore be approximated by triangle mesh surfaces.

Polygons are rendered either with fill or in outline.

glRect* draws axis-aligned rectangles in the $z = 0$ plane for two specified corners.

Curves and curved surfaces are rendered as polygonal curves or triangle-mesh surfaces with enough pieces to appear smooth.
State changes are made by the functions

```c
void glEnable(GLenum capability);
void glDisable(GLenum capability);
```

where capability values include the following.

- GL_BLEND: blend foreground and background RGBA values
- GL_CULL_FACE: cull back faces
- GL_FOG: simulate fog
- GL_LINE_STIPPLE: create stippled lines
- GL_LIGHTING: model reflected light

Current state can be queried by glIsEnabled, glGetBooleanV, etc.
void glPointSize(GLfloat size);
sets point width in pixels to round(size) > 0 with default value 1. By default there is no antialiasing, resulting in square points.

void glLineWidth(GLfloat w);
sets line width to \( w > 0 \), with default value 1, measured in the \( x \) or \( y \) direction, depending on the slope. Antialiasing allows non-integer width.

void glPolygonMode(GLenum face, GLenum mode);

face = GL_FRONT, GL_BACK, or GL_FRONT_AND_BACK (default)
mode = GL_POINT, GL_LINE, or GL_FILL (default)

Note that shared vertices or edges are drawn only once so that, in the case of blending, the intensity is not affected. Also, an orientable surface must be modeled by consistently-oriented polygons.
Normal Vectors

- Normal vectors are used by lighting calculations to determine angles of incidence, intensities, and directions of reflection.
- A normal vector is associated with each vertex (even though the normal direction is not defined at a vertex).
- glNormal*() sets the normal for all subsequently defined vertices within a glBegin/glEnd pair.
- Normal vectors must have unit length unless automatic normalization is enabled (as it must be if the modelview matrix includes scaling).
- The normal directions must be consistent; e.g., all outward in the case of a closed surface.
Vertex arrays serve two purposes.

1. They reduce the number of function calls (which have high overhead on some systems).

2. They eliminate redundant processing of shared vertices. A cube, for example, has 8 vertices, each shared by 3 faces, corresponding to 24 vertices if defined as 6 polygons.

Vertex arrays cannot be included in display lists because they remain on the client side unless included in a buffer object. Use of vertex arrays involves three steps.

1. Activate (enable) up to 8 arrays: vertex coordinates, surface normals, RGBA colors, secondary colors, color indices, fog coordinates, texture coordinates, or polygon edge flags:

```
void glEnableClientState(GLenum array);
```

— one call for each enabled array.
Specify array data using the functions defined in Table 2-4; e.g.,

```c
void glVertexPointer(GLint size, GLenum type, GLsizei stride, const GLvoid *pointer);
```

- `size` = 2, 3, or 4 (number of coordinates)
- `type` = `GL_SHORT`, `GL_INT`, `GL_FLOAT`, or `GL_DOUBLE`
- `stride` = byte offset between vertices, or 0 if tightly packed

Dereference the pointers, and transfer the data to the server. The following function gets the indexed data values in all currently enabled arrays.

```c
void glDrawElements(GLenum mode, GLsizei count, GLenum type, const GLvoid *indices);
```

- `mode` = argument for `glBegin` (`GL_POLYGON`, `GL_LINES`, etc)
- `count` = Number of elements
- `type` = type of indices (`GL_UNSIGNED_BYTE`, `GL_UNSIGNED_SHORT`, or `GL_UNSIGNED_INT`)
Buffer objects allow explicit specification of which data is to be stored in the server, possibly avoiding redundant transfers of data that has not changed. Vertex arrays are a good candidate for buffer objects. Pixel data can also be stored in buffer objects.

Attribute groups are predefined collections of related state variables which can be saved and restored on a stack with a single command.
Following are some general rules to keep in mind when constructing a polygonal surface.

- Keep polygon orientation (winding) consistent; e.g., CCW-ordered vertices as viewed from one side.
- Use only triangles for curved surfaces.
- Use a finer grid on nearer objects, high-curvature surfaces, and boundary curves.
- Avoid T intersections. A T intersection is a vertex of one polygon in the interior of an edge of another polygon. In a triangle mesh surface, the intersection of a pair of triangles, if not empty, is a shared vertex or a shared edge.
T Intersection: Fig 2-16

Undesirable

OK
1. Geometric objects are created from geometric primitives with smooth shading. What does that mean?
Review Questions

1. Geometric objects are created from geometric primitives with smooth shading. What does that mean?

2. What does it mean to clear the window? When and how do we do it?
1. Geometric objects are created from geometric primitives with smooth shading. What does that mean?
2. What does it mean to clear the window? When and how do we do it?
3. What are the buffers associated with a window?
1. Geometric objects are created from geometric primitives with smooth shading. What does that mean?
2. What does it mean to clear the window? When and how do we do it?
3. What are the buffers associated with a window?
4. What is indexed color?
Review Questions

1. Geometric objects are created from geometric primitives with smooth shading. What does that mean?
2. What does it mean to clear the window? When and how do we do it?
3. What are the buffers associated with a window?
4. What is indexed color?
5. What is the alpha channel?
6. How do we change the current drawing color?
7. When is the current color irrelevant?
8. Why and how do we force completion of drawing?
9. What is a viewport, and how do we create one?
10. What is a view volume, and how do we create one?
11. How do we achieve exact rasterization?
12. What are floating-point numbers?
13. How do we specify vertices?
14. What is an OpenGL polygon, and what are the options for rendering one?
Review Questions

1. Geometric objects are created from geometric primitives with smooth shading. What does that mean?
2. What does it mean to clear the window? When and how do we do it?
3. What are the buffers associated with a window?
4. What is indexed color?
5. What is the alpha channel?
6. How do we change the current drawing color?
Review Questions

1. Geometric objects are created from geometric primitives with smooth shading. What does that mean?
2. What does it mean to clear the window? When and how do we do it?
3. What are the buffers associated with a window?
4. What is indexed color?
5. What is the alpha channel?
6. How do we change the current drawing color?
7. When is the current color irrelevant?
8. Why and how do we force completion of drawing?
9. What is a viewport, and how do we create one?
10. What is a view volume, and how do we create one?
11. How do we achieve exact rasterization?
12. What are floating-point numbers?
13. How do we specify vertices?
14. What is an OpenGL polygon, and what are the options for rendering one?
1. Geometric objects are created from geometric primitives with smooth shading. What does that mean?
2. What does it mean to clear the window? When and how do we do it?
3. What are the buffers associated with a window?
4. What is indexed color?
5. What is the alpha channel?
6. How do we change the current drawing color?
7. When is the current color irrelevant?
8. Why and how do we force completion of drawing?
Review Questions

1. Geometric objects are created from geometric primitives with smooth shading. What does that mean?
2. What does it mean to clear the window? When and how do we do it?
3. What are the buffers associated with a window?
4. What is indexed color?
5. What is the alpha channel?
6. How do we change the current drawing color?
7. When is the current color irrelevant?
8. Why and how do we force completion of drawing?
9. What is a viewport, and how do we create one?
1. Geometric objects are created from geometric primitives with smooth shading. What does that mean?

2. What does it mean to clear the window? When and how do we do it?

3. What are the buffers associated with a window?

4. What is indexed color?

5. What is the alpha channel?

6. How do we change the current drawing color?

7. When is the current color irrelevant?

8. Why and how do we force completion of drawing?

9. What is a viewport, and how do we create one?

10. What is a view volume, and how do we create one?
1. Geometric objects are created from geometric primitives with smooth shading. What does that mean?
2. What does it mean to clear the window? When and how do we do it?
3. What are the buffers associated with a window?
4. What is indexed color?
5. What is the alpha channel?
6. How do we change the current drawing color?
7. When is the current color irrelevant?
8. Why and how do we force completion of drawing?
9. What is a viewport, and how do we create one?
10. What is a view volume, and how do we create one?
11. How do we achieve exact rasterization?
Review Questions

1. Geometric objects are created from geometric primitives with smooth shading. What does that mean?
2. What does it mean to clear the window? When and how do we do it?
3. What are the buffers associated with a window?
4. What is indexed color?
5. What is the alpha channel?
6. How do we change the current drawing color?
7. When is the current color irrelevant?
8. Why and how do we force completion of drawing?
9. What is a viewport, and how do we create one?
10. What is a view volume, and how do we create one?
11. How do we achieve exact rasterization?
12. What are floating-point numbers?
Review Questions

1. Geometric objects are created from geometric primitives with smooth shading. What does that mean?
2. What does it mean to clear the window? When and how do we do it?
3. What are the buffers associated with a window?
4. What is indexed color?
5. What is the alpha channel?
6. How do we change the current drawing color?
7. When is the current color irrelevant?
8. Why and how do we force completion of drawing?
9. What is a viewport, and how do we create one?
10. What is a view volume, and how do we create one?
11. How do we achieve exact rasterization?
12. What are floating-point numbers?
13. How do we specify vertices?
Review Questions

1. Geometric objects are created from geometric primitives with smooth shading. What does that mean?
2. What does it mean to clear the window? When and how do we do it?
3. What are the buffers associated with a window?
4. What is indexed color?
5. What is the alpha channel?
6. How do we change the current drawing color?
7. When is the current color irrelevant?
8. Why and how do we force completion of drawing?
9. What is a viewport, and how do we create one?
10. What is a view volume, and how do we create one?
11. How do we achieve exact rasterization?
12. What are floating-point numbers?
13. How do we specify vertices?
14. What is an OpenGL polygon, and what are the options for rendering one?