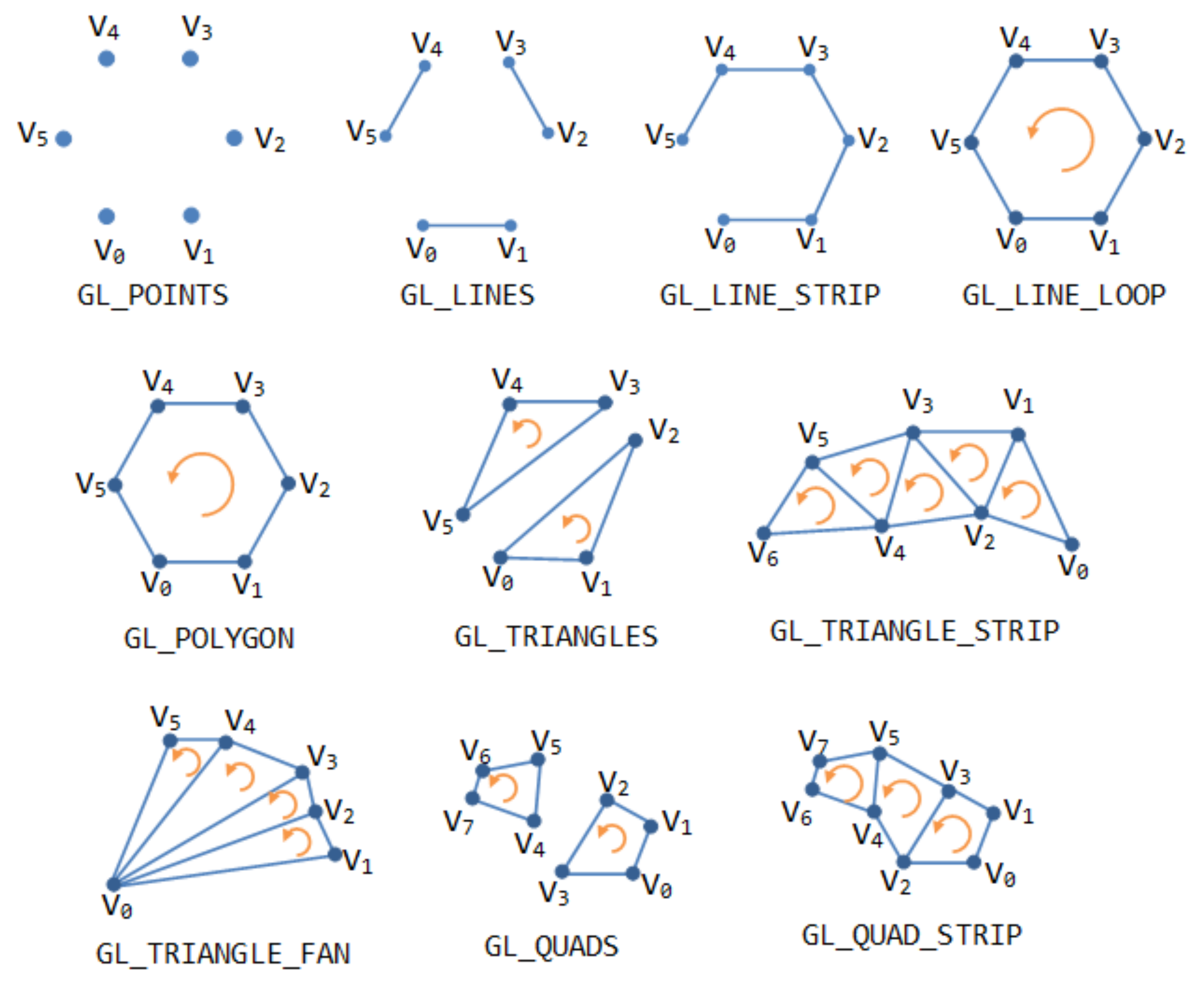
Graphics Foundations

Part 4

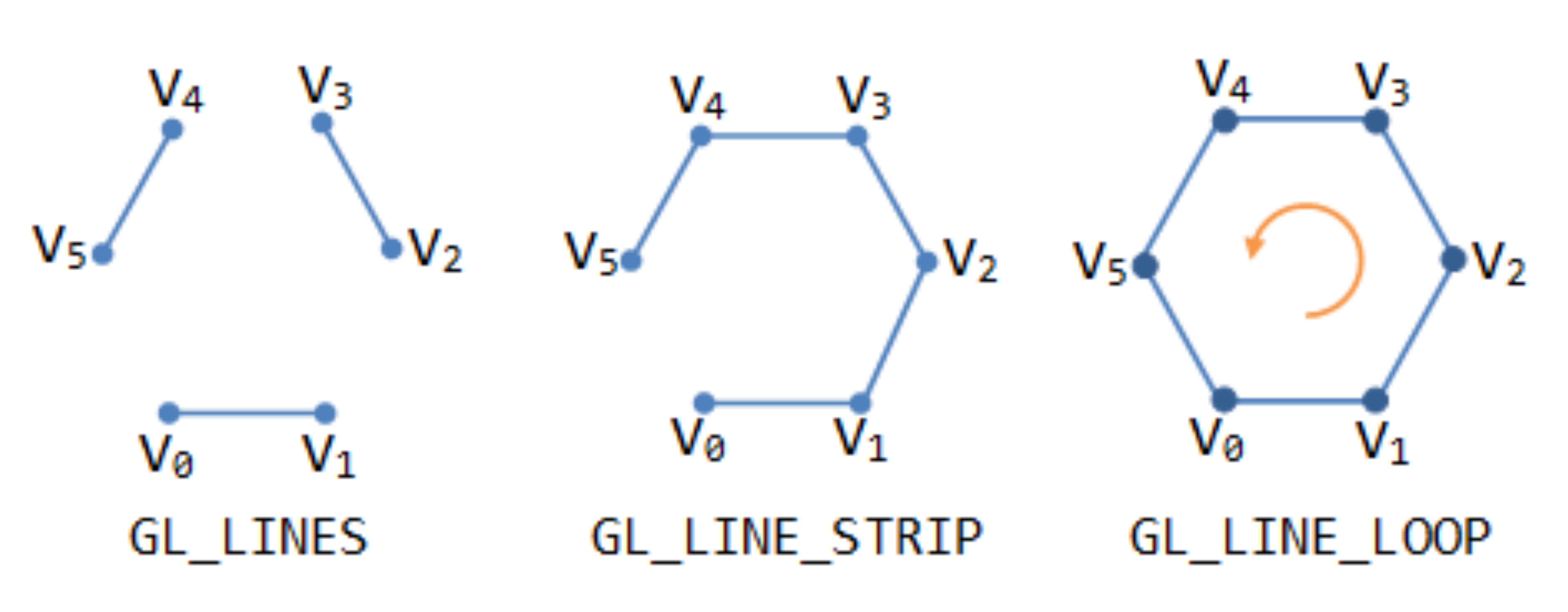
Drawing different types.



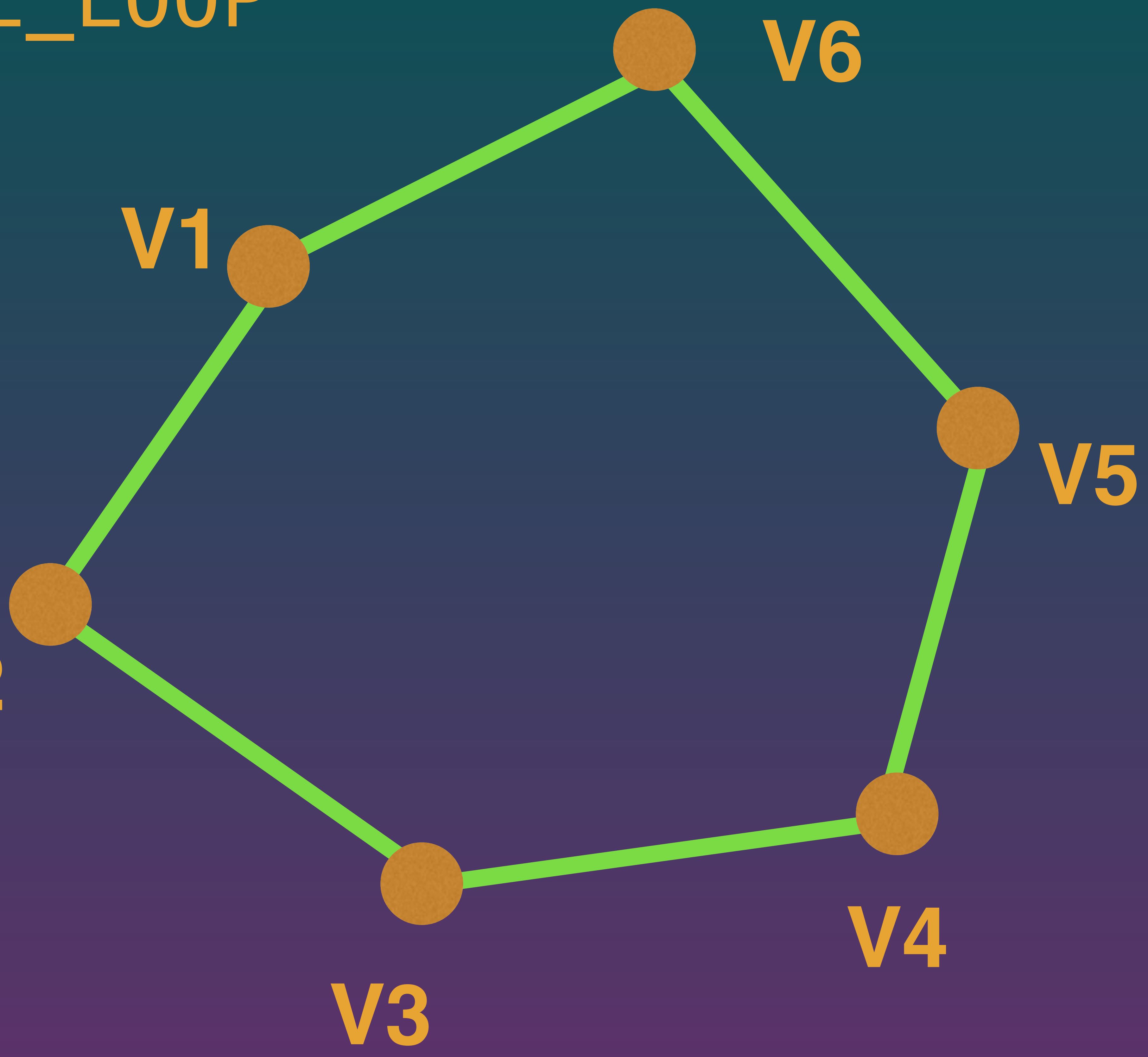
OpenGL Primitives

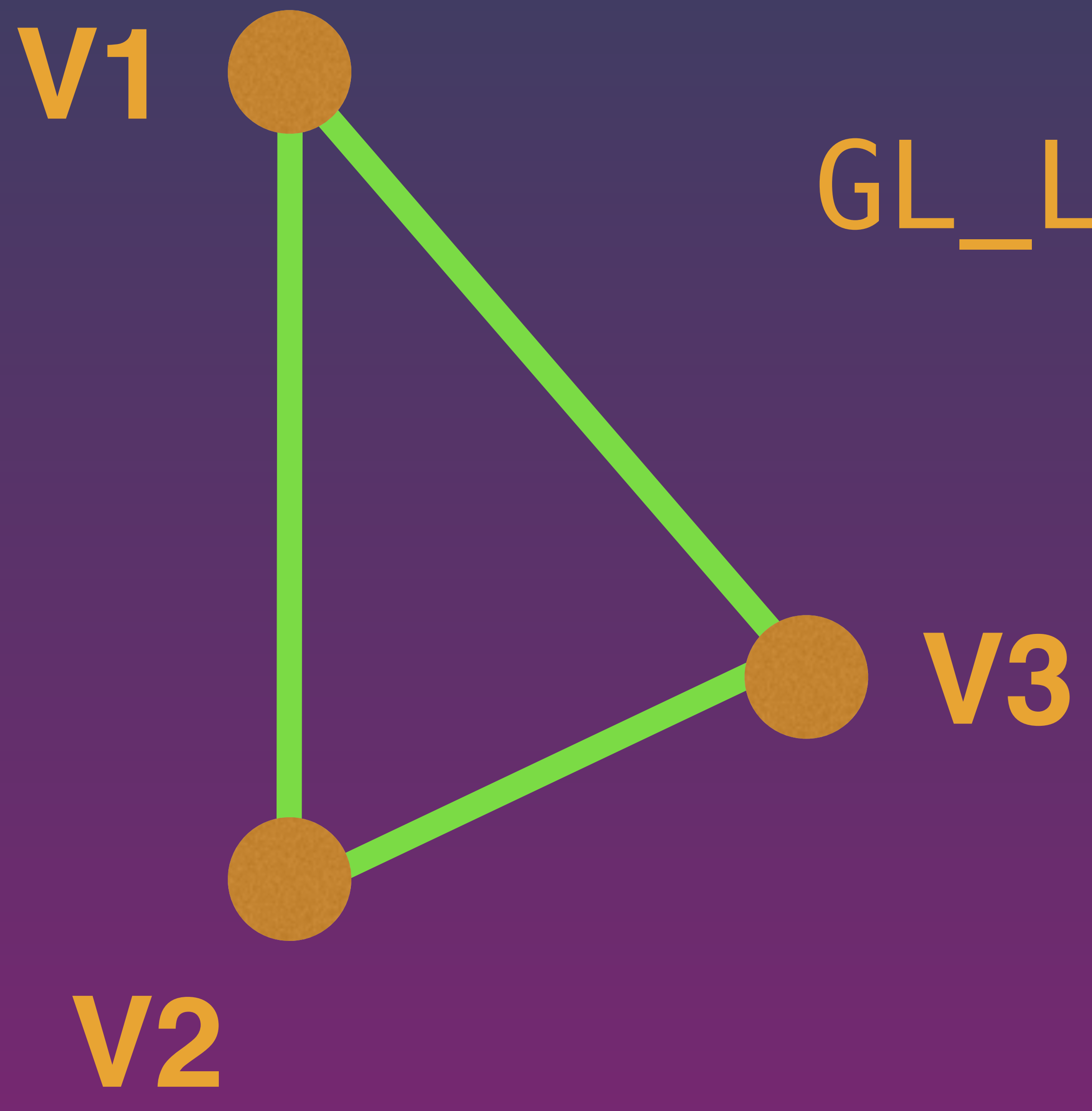
Drawing lines in OpenGL.

```
glDrawArrays(GL_LINES, 0, points.size());
```



GL_LINE_LOOP



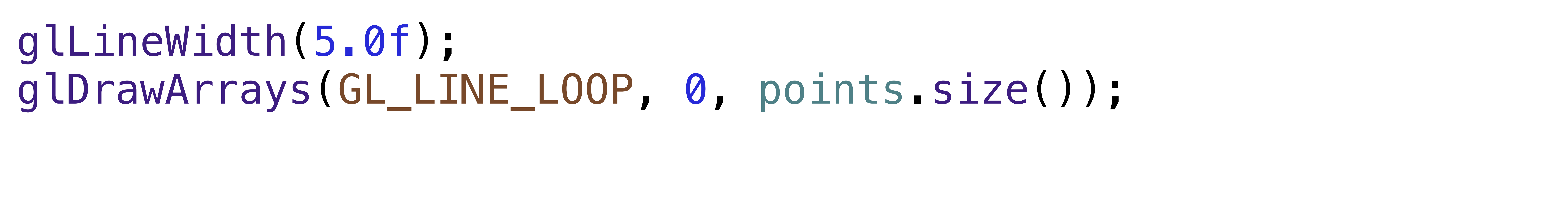


GL_LINE_LOOP

Changing line width.

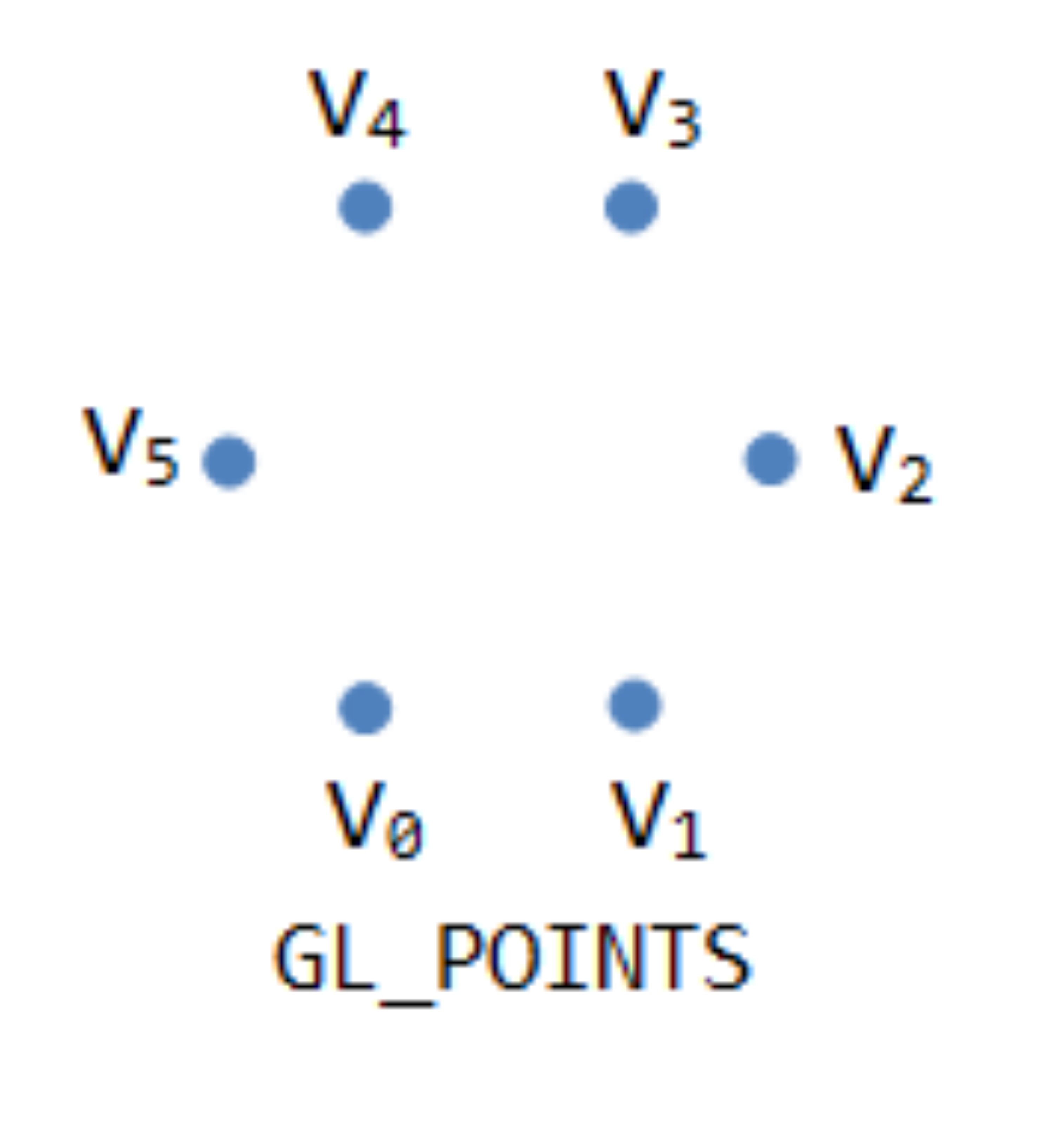
void glLineWidth (GLfloat width);

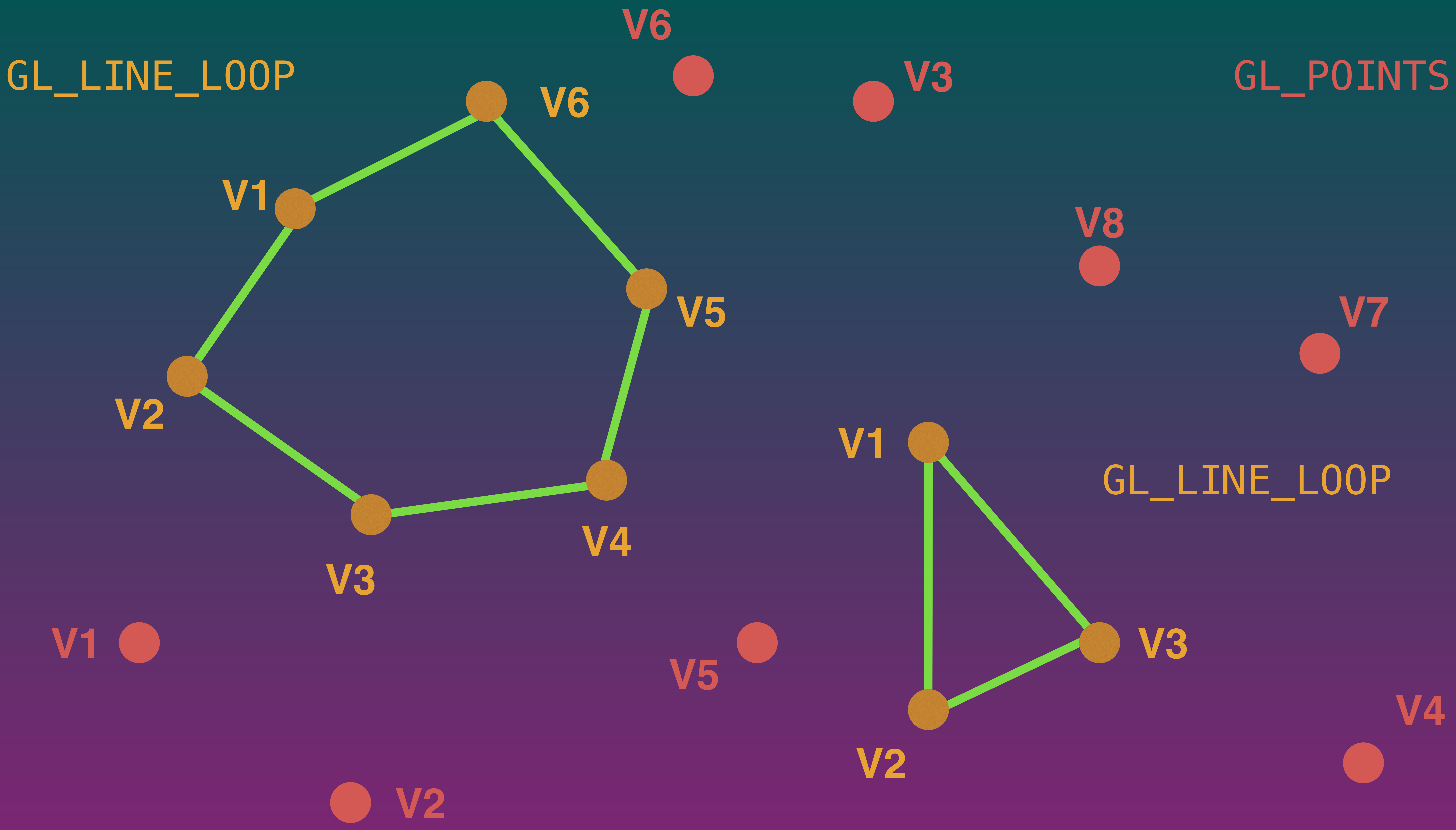
Changes OpenGL line width (in pixels).



```
glLineWidth(5.0f);
glDrawArrays(GL_LINE_LOOP, 0, points.size());
```

Drawing points in OpenGL.


V_4
V_3
V_5
V_2
V_0
V_1
GL_POINTS



GL_LINE_LOOP

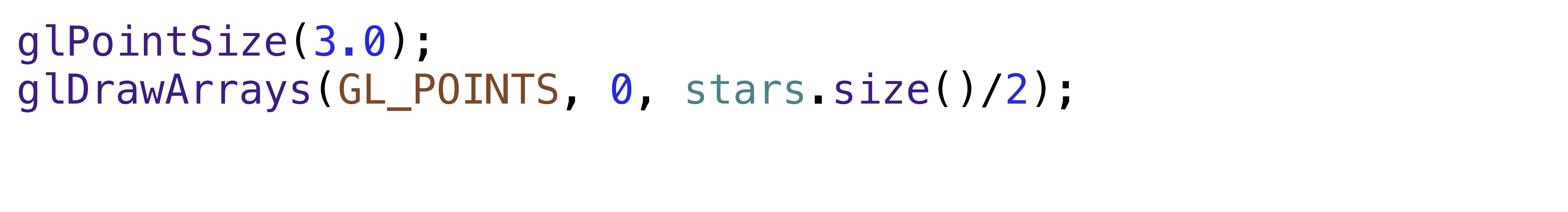
GL_POINTS

GL_LINE_LOOP

Changing point size.

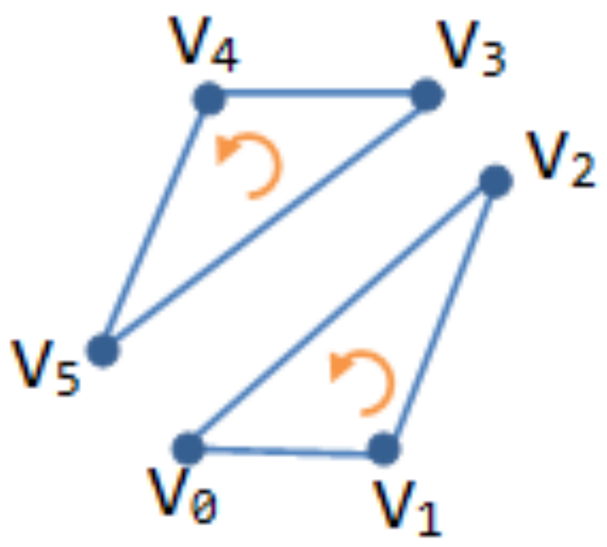
`void glPointSize (GLfloat size);`

Changes OpenGL points size (in pixels).

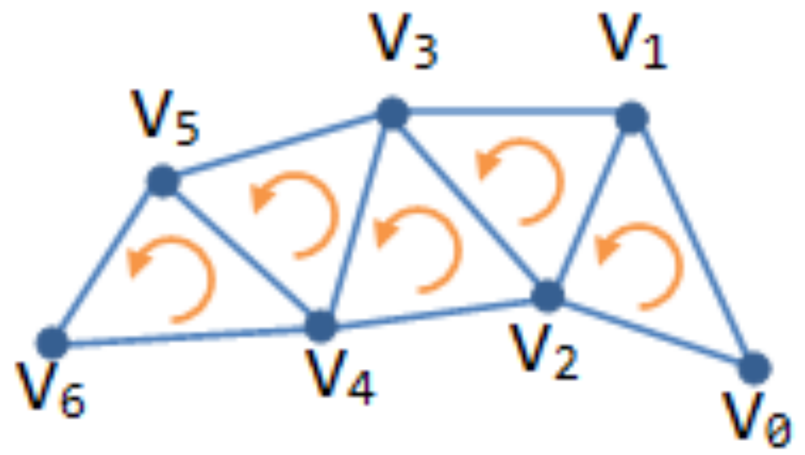


```
glPointSize(3.0);
glDrawArrays(GL_POINTS, 0, stars.size()/2);
```

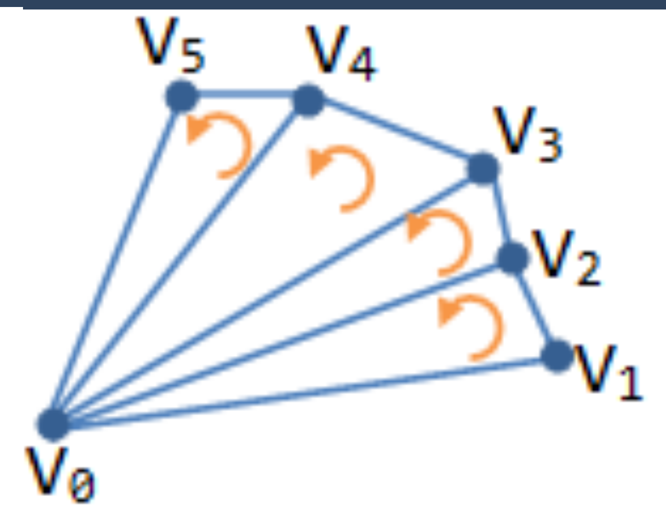
Drawing triangles.



GL_TRIANGLES

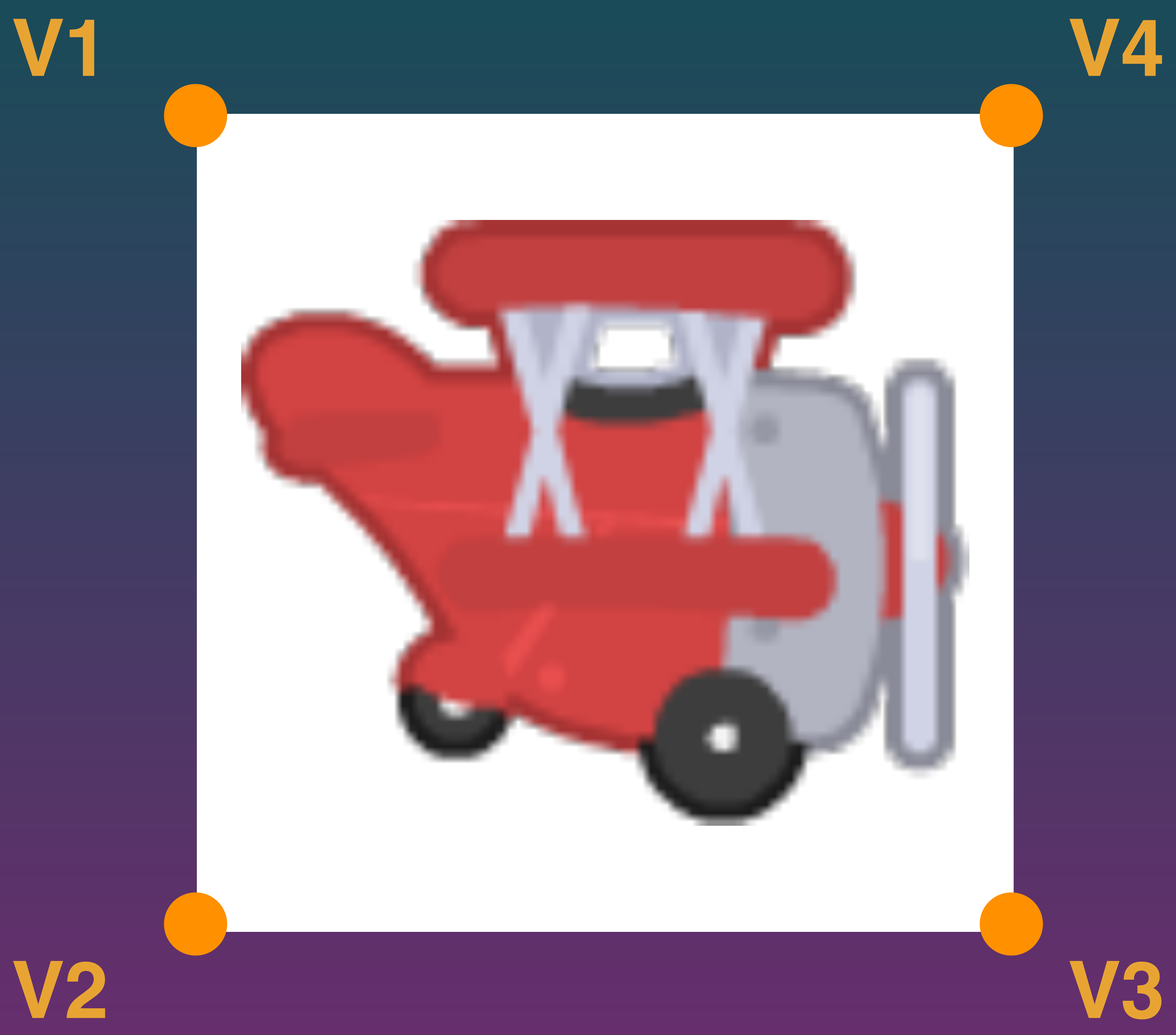


GL_TRIANGLE_STRIP

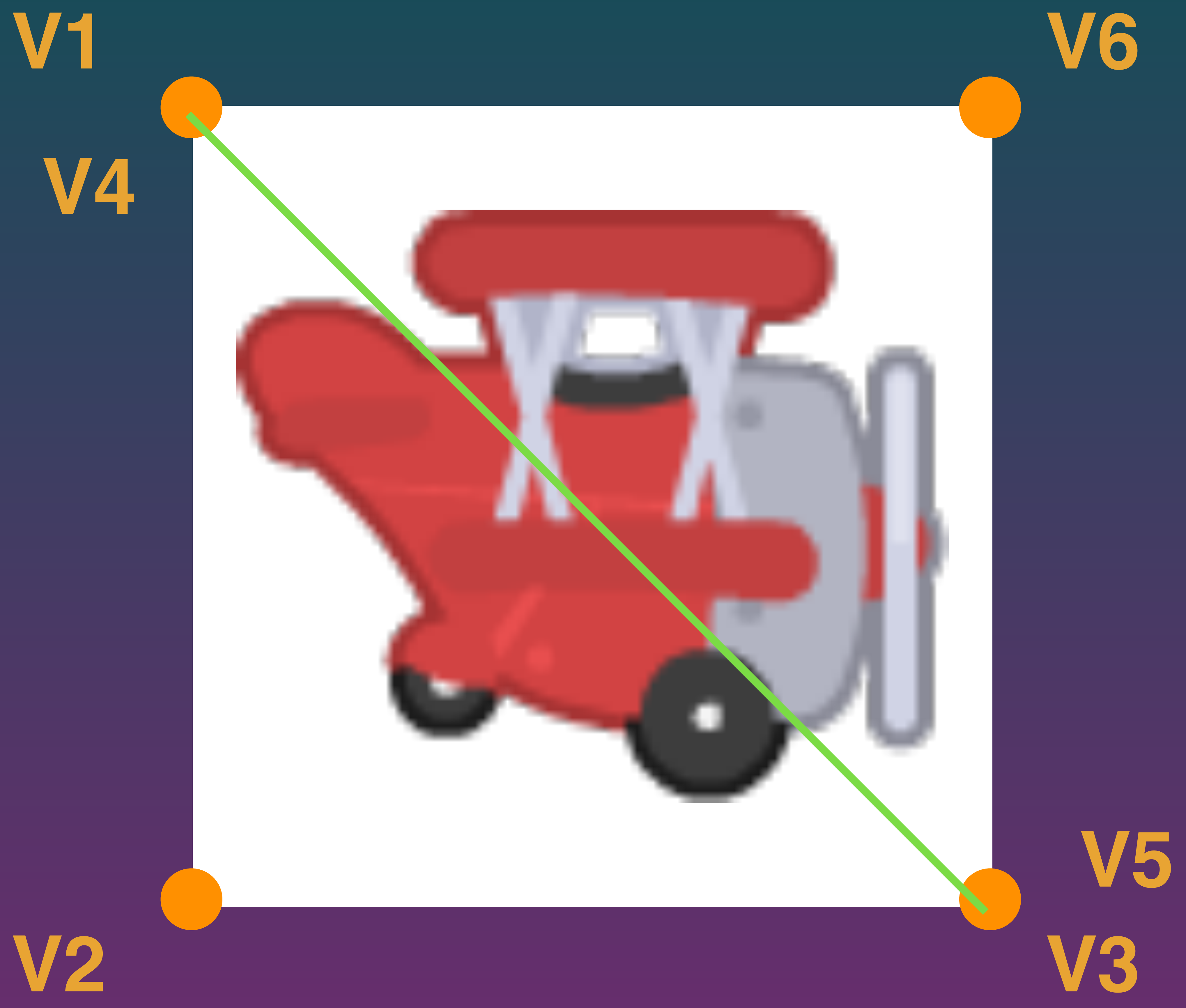


GL_TRIANGLE_FAN

GL_QUADS

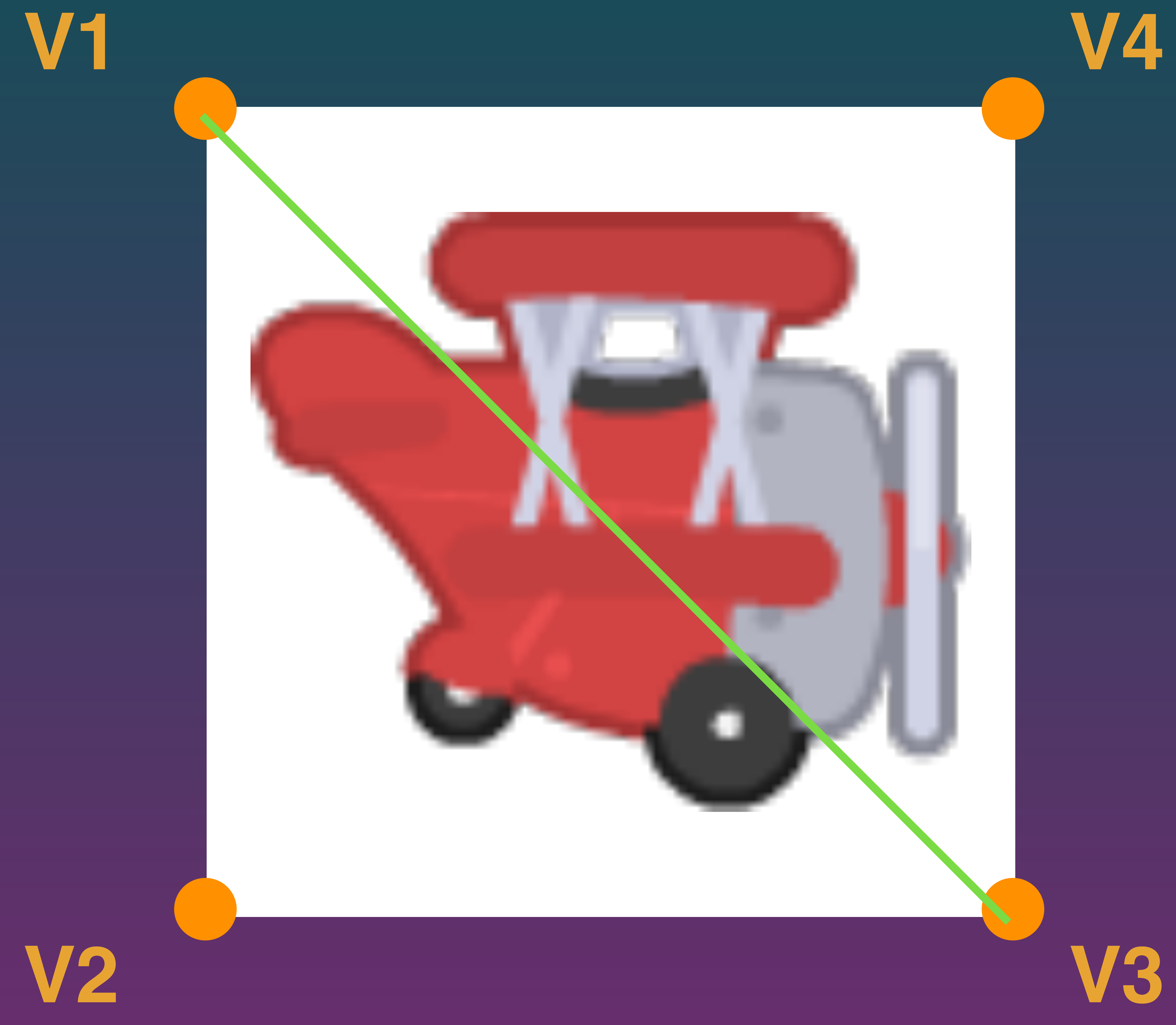


GL_TRIANGLES



Drawing using indexes.

GL_TRIANGLES



V1

V4

V2

V3

V1,V2,V3 V1,V3,V4

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

Draws vertices defined by glVertexPointer using a list of indices from that array.

```
std::vector<unsigned int> indices = {0,1,2,0,2,3};
glDrawElements(GL_TRIANGLES, 6, GL_UNSIGNED_INT, indices.data());
```

Vertex Buffer Objects (VBOs).

Vertex Buffer Objects are vertex arrays stored on the GPU, so you can draw really large vertex arrays really quickly.

Perfect for large static geometry like levels.

Creating VBOs

```
void glGenBuffers (GLsizei n, GLuint *buffers);
```

Generates new buffers.

```
glGenBuffers(1, &myVertexBuffer);
```

glBindBuffer (GLenum target, GLuint buffer);

Binds a buffer. Target must be GL_ARRAY_BUFFER for vertex array buffers.

```
glGenBuffers(1, &myVertexBuffer);

glBindBuffer(GL_ARRAY_BUFFER, myVertexBuffer);
```

```
void glBufferData (GLenum target, GLsizeiptr size, const GLvoid *data,
GLenum usage);
```

Sets the buffer's data. Target must be GL_ARRAY_BUFFER for vertex array buffers. Usage must be GL_STATIC_DRAW for static buffers.

ATTENTION: THE SIZE IS IN BYTES, SO IT'S THE SIZE OF THE ARRAY * SIZE OF EACH ARRAY ELEMENT!

```
glGenBuffers(1, &myVertexBuffer);

glBindBuffer(GL_ARRAY_BUFFER, myVertexBuffer);

glBufferData(GL_ARRAY_BUFFER, vertexData.size() * sizeof(float),
vertexData.data(), GL_STATIC_DRAW);
```

VBOs work like regular vertex arrays, so you'll need one for each vertex attribute, or mix them in one array.

```
// somewhere in your class
GLuint myVertexBuffer;

// initialize the buffer. THIS IS ONLY DONE ONCE!
glGenBuffers(1, &myVertexBuffer);
glBindBuffer(GL_ARRAY_BUFFER, myVertexBuffer);
glBufferData(GL_ARRAY_BUFFER, vertexData.size() * sizeof(float), vertexData.data(),
GL_STATIC_DRAW);
```

Drawing VBOs

Drawing done the same as before, only…

- We bind a buffer using glBindBuffer before calling glVertexPointer, glTexCoordPointer or glColorPointer.

- We pass NULL as a pointer to glVertexPointer, glTexCoordPointer or glColorPointer

- We must unbind our buffer after we are done (bind buffer 0).

VBOs work like regular vertex arrays, so you'll need one for each vertex attribute, or mix them in one array.

```
// somewhere in your class
GLuint myVertexBuffer;

// initialize the buffer. THIS IS ONLY DONE ONCE!
glGenBuffers(1, &myVertexBuffer);
glBindBuffer(GL_ARRAY_BUFFER, myVertexBuffer);
glBufferData(GL_ARRAY_BUFFER, vertexData.size() * sizeof(float), vertexData.data(),
GL_STATIC_DRAW);

// in our render function

glEnableClientState(GL_VERTEX_ARRAY);

glBindBuffer(GL_ARRAY_BUFFER, myVertexBuffer);
glVertexPointer(2, GL_FLOAT, 0, NULL);
glBindBuffer(GL_ARRAY_BUFFER, 0);

glDrawArrays(GL_QUADS, 0, numVertices);
```

Destroying VBOs

```
void glDeleteBuffers (GLsizei n, const GLuint *buffers);
```

Deletes an existing buffer.

```
glDeleteBuffers(1, &myVertexBuffer);
```

Index VBOs

Must use the GL_ELEMENT_ARRAY_BUFFER target.

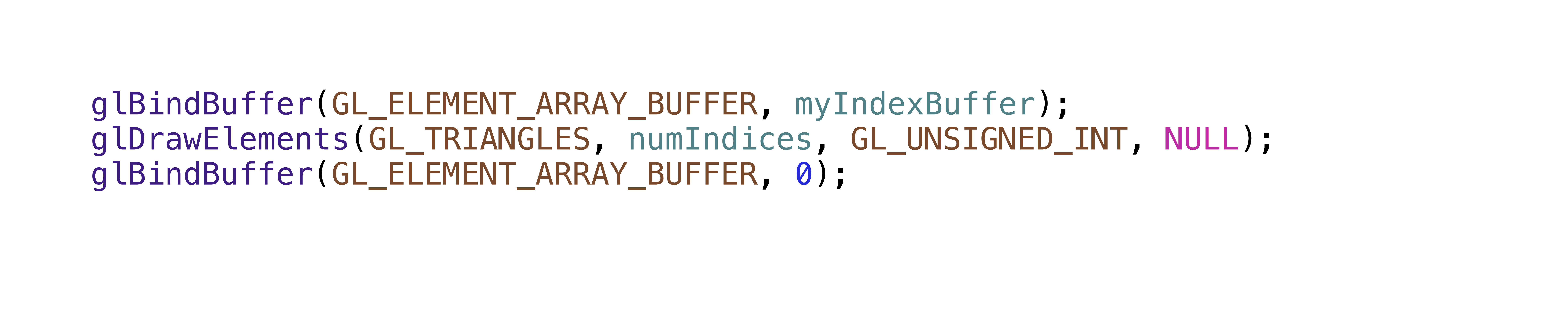
Creating.

```
glGenBuffers(1, &myIndexBuffer);

glBindBuffer(GL_ELEMENT_ARRAY_BUFFER, myIndexBuffer);

glBufferData(GL_ELEMENT_ARRAY_BUFFER, indexData.size() *
sizeof(unsigned int), indexData.data(), GL_STATIC_DRAW);
```

Drawing



```
glBindBuffer(GL_ELEMENT_ARRAY_BUFFER, myIndexBuffer);
glDrawElements(GL_TRIANGLES, numIndices, GL_UNSIGNED_INT, NULL);
glBindBuffer(GL_ELEMENT_ARRAY_BUFFER, 0);
```

View culling.

Not drawing things that are outside of the screen.

Need to check if things are outside of our view rectangle. (camera.x -ortho_width, camera.x + ortho_width, camera.y +ortho_height, camera.y - ortho.height)

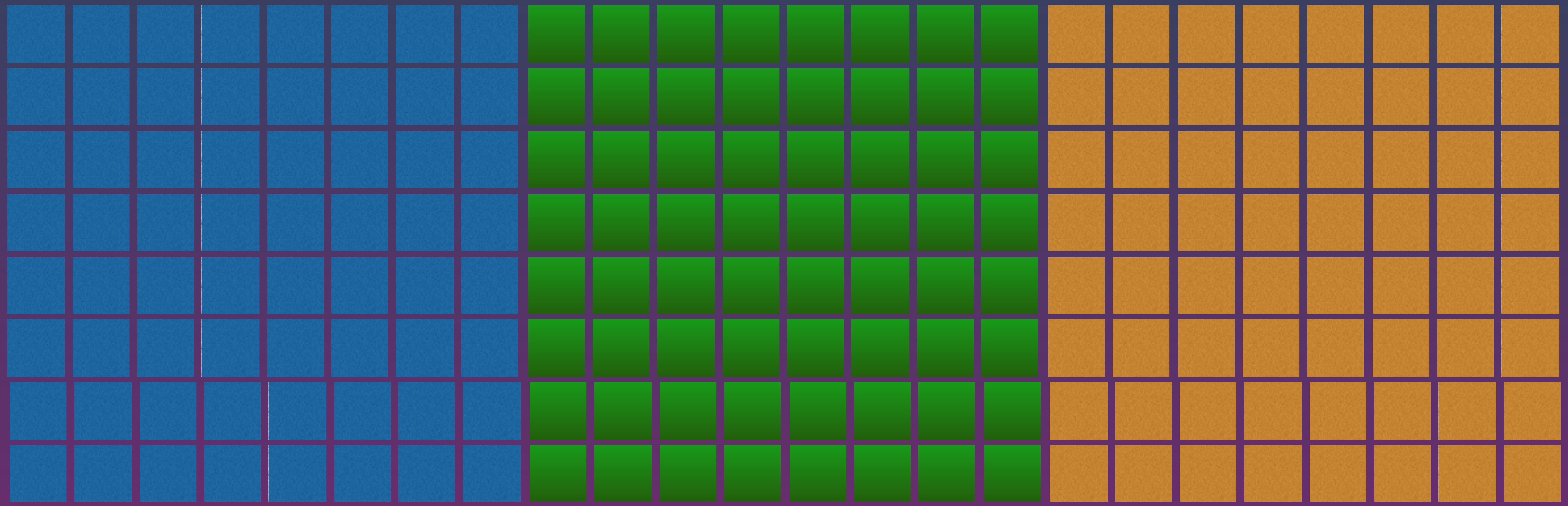
Culling entities.

If we don’t have rotation, we can
just check if the entity’s x +- width/2 and y +- height/2 is within the view rectangle

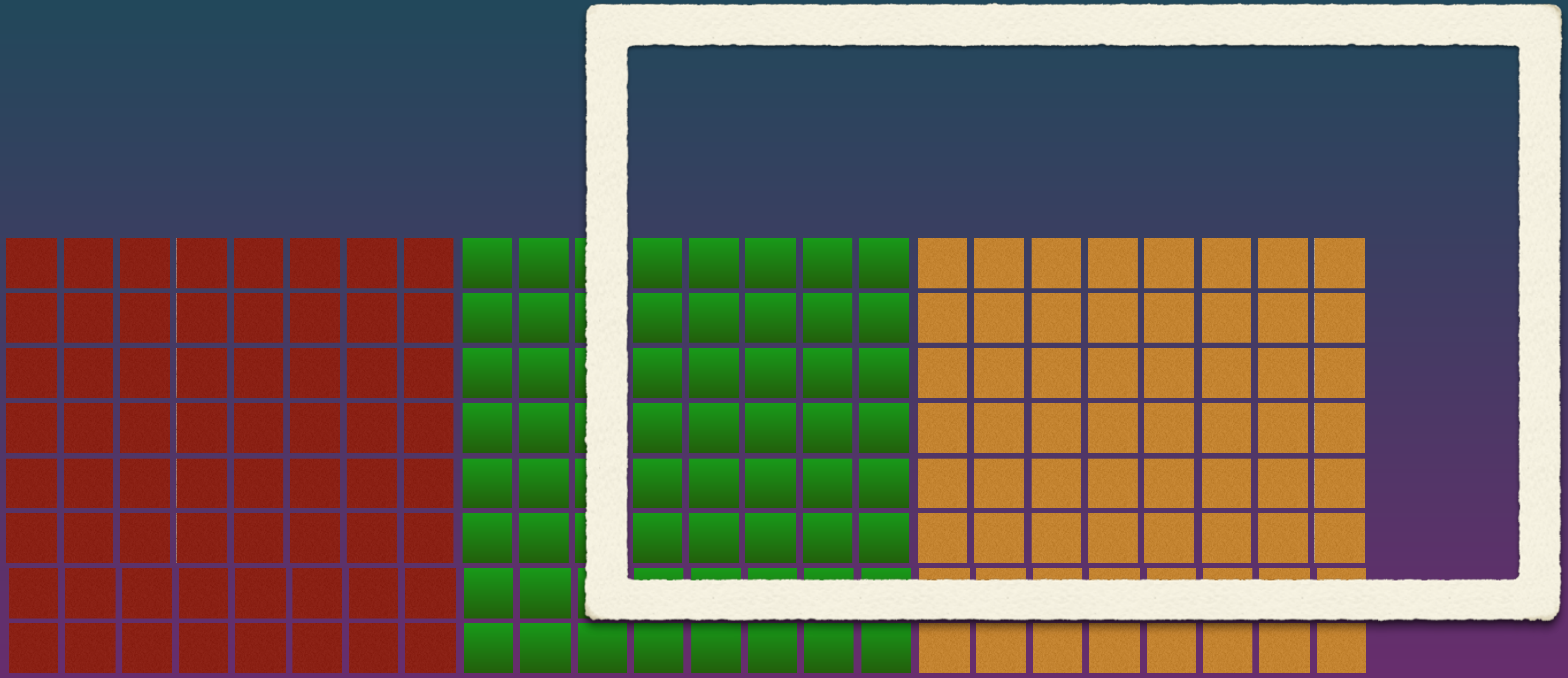
If we do have rotation, we can assign a radius to each entity based on its largest dimension and check if it's within our view rectangle.

Culling tilemaps.

You can break up your level tiles into smaller chunks with their own VBO so we don't have to draw the entire level every time.



When you render, figure out which chunks of the level are visible and only render the visible ones.



Video time!

Final assignment details.

What is required?

- Must have a title screen and proper states for game over, etc.
- Must have a way to quit the game.
- Must have music and sound effects.
- Must have at least 3 different levels or be procedurally generated.
- Must be either local multiplayer or have AI (or both!).
- Must have at least some animation or particle effects.

Bonus points for…

- Using OpenGL ES2 standards (matrices + shaders + triangles).
- Having 3D elements.
- Having shader effects.

(we haven't covered any of this yet!)