COMP4610/COMP6461

Week 6 - Physics and Introduction to 3D Graphics

<Print version>

Admin

[Feedback for Labs]

- Lab feedback is provided via Wattle.
- This includes individual marks for the three tasks, and comments.
- If you would like additional feedback please discuss with your tutor.
- Feedback will be provided on Tuesday week, (i.e. 6 working days later), ready for the first lab.

[A few Small Changes to the Labs]

- Future labs / assignments have been renamed to **comp4610-xxx** (to avoid collisions).
- There is no longer any need to add me as a developer. This is done automatically when you fork (you should see a marker user added automatically).
- We now send out a confirmation email receipt at 7 PM letting you know your work was received.

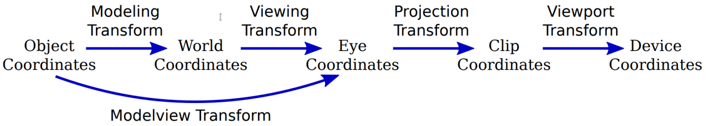
[Assignment 1]

- First Assignment is due at the end of this week
- Remember to follow the instructions on the spec.
- Your submission should include a 5-minute **video** demonstrating your solution.
- Videos should be submitted as part of your repo. Just make sure they are $<$100MB.
- The assignment spec makes mention of being able to use **JUnit** for testing. If you would like to do this, feel free. I've added the JARs back into the repo.

3D Viewing Transformations

3D Viewing

- Viewing processes for a 3D scene are in many ways similar to that of 2D. However, the extra dimension brings with it a host of complexities such as lighting and viewing projections.
- Coordinates undergo a series of transformations to produce the final image on the screen.



Introduction to Computer Graphics (D.J. Eck) Ch 3.3

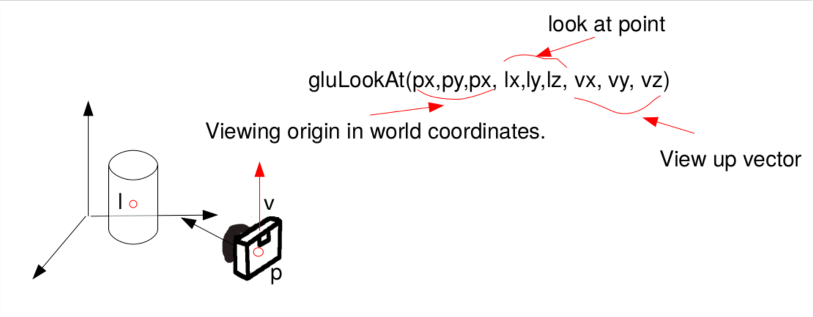
3D Viewing

This means we need to...

1. Apply a **model** transform (position rotate and scale the object into the world)
2. Apply a **view** transform (transform the view based on the camera's location and rotation)
3. Apply a **projection** transform (project the 3d scene onto the 2d viewing plane.

- In OpenGL [1,2] are handled by the **MODELVIEW** matrix, while [3] is handled by the **PROJECTION** matrix.
- In general, [1,2] change every frame, while [3] (almost) never changes.

Viewing Transformation



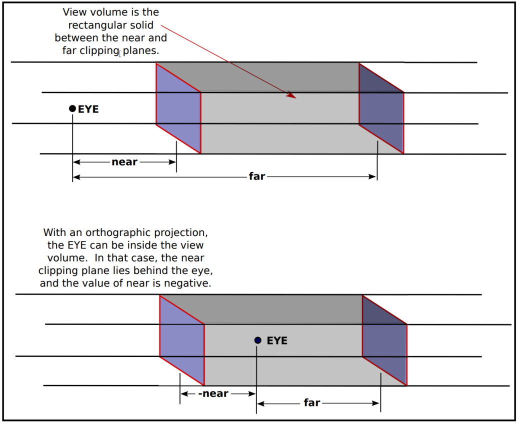
Projection

- Orthogonal projections are parallel projections in which objects appear the same size as their distance from the viewer changes.
- The view volume (**frustrum**) forms a hyper rectangle (cuboid).
- **glOrtho()** will turn viewing coordinates into normalised projected coordinates. These normalised coordinates are a cube of side length 2 centred on the origin.

In OpenGL

```
glu.gluOrtho2D(0.0, dim.getWidth(), 0.0, dim.getHeight(), ); // option 1
gl.glOrtho2D(0.0, dim.getWidth(), 0.0, dim.getHeight(), −1, 1 ); // option 2
```

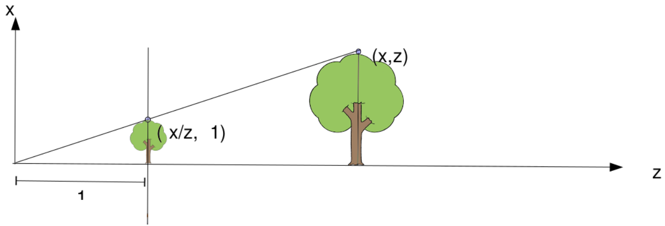
Orthogonal Projection



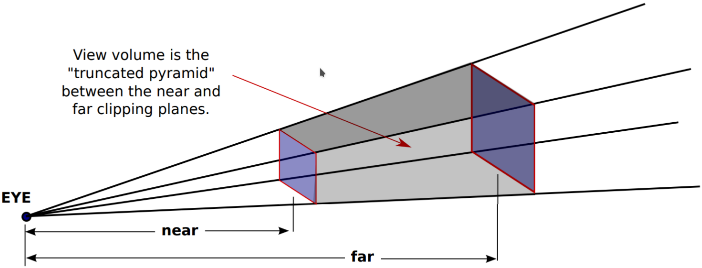
Introduction to Computer Graphics (D.J. Eck) Ch 3.3

- Projection requires dividing by z.
- A homogeneous matrix can be used to describe a perspective transformation.
- There are two functions that can be used in **OpenGL** to give a perspective projection. They are:

```
gluPerspective(theta, aspect, dnear, dfar);
glFrustum(xwmin, xwmax, ywmin, yzmax, dnear, dfar);
```



Perspective Projection



Introduction to Computer Graphics (D.J. Eck) Ch 3.3

Perspective Projection

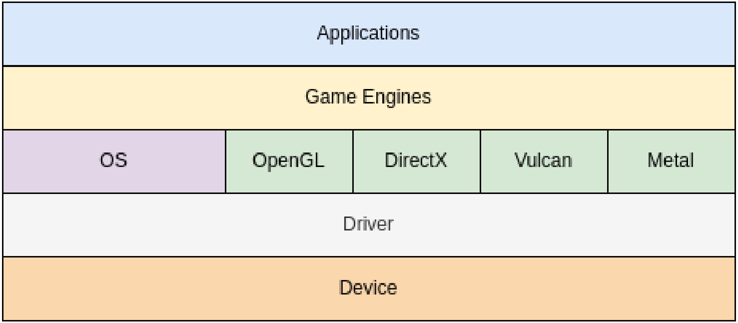
Perspective projection requires a division in the calculation. This can be achieved by using the division that is done when homogeneous points are converted into cartesian points. So the matrix used by **OpenGL** for perspective projection glFrustrum(l,r,b,t,n,f) is

$$\begin{bmatrix} \frac{2n}{r-l} & 0 & \frac{r+l}{r-l} & 0 \\ 0 & \frac{2n}{t-b} & \frac{t+b}{t-b} & 0 \\ 0 & 0 & -\frac{f+n}{f-n} & \frac{2fn}{f-n} \\ 0 & 0 & -1 & 0 \end{bmatrix}$$

Break

OpenGL

Graphics APIs



OpenGL in more detail

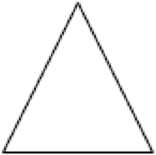
- **OpenGL 1.1**
    - **Fixed function** pipeline.
    - Devices typically implemented commands in hardware.
    - Fairly good software fallback.
    - **Easy** to **setup**, **hard** to **customize**.
- **OpenGL 2.0**
    - **Programmable pipeline** via **shaders**.
    - Still has support for fixed function pipeline (via emulation).
    - This is what we typically use in this course.
    - **Harder** to **setup**, **easier** to **customize**.
- **OpenGL ES**
    - The embedded systems subset of OpenGL.
    - Drops fixed-function support.
    - Probably the most widely deployed 3D graphics API in history.
    - WebGL is based on this.

OpenGL

OpenGL as a state machine

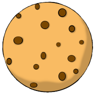
Allegro (a mostly stateless graphics library)

```
void al_draw_triangle(float x1, float y1, float x2, float y2
    , float x3, float y3, ALLEGRO_COLOR color, float
    thickness)
```



```
void al_draw_filled_triangle(float x1, float y1, float x2,
    float y2, float x3, float y3, ALLEGRO_COLOR color)
```

```
int al_draw_prim(const void* vtxs, const ALLEGRO_VERTEX_DECL
    * decl, ALLEGRO_BITMAP* texture, int start, int end, int
    type)
```

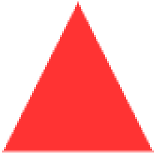


OpenGL (a stateful graphics library)

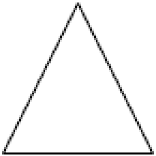
```
gl.glBegin(GL2.GL_POLYGON);
gl.glVertex2d(0.0, 0.0);
gl.glVertex2d(50.0, 100.0);
gl.glVertex2d(-50.0, 100.0);
gl.glEnd();
```

OpenGL (a stateful graphics library)

```
gl.glBegin(GL2.GL_POLYGON);
gl.glColor3f(1.0, 0.0, 0.0);
gl.glVertex2d(0.0, 0.0);
gl.glVertex2d(50.0, 100.0);
gl.glVertex2d(-50.0, 100.0);
gl.glEnd();
```

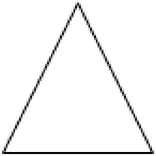


```
gl.glPolygonMode(GL_FRONT_AND_BACK, GL_LINE);
gl.glBegin(GL2.GL_POLYGON);
gl.glVertex2d(0.0, 0.0);
gl.glVertex2d(50.0, 100.0);
gl.glVertex2d(-50.0, 100.0);
gl.glEnd();
```

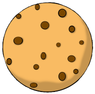


OpenGL (a stateful graphics library)

```
gl.glPolygonMode(GL_FRONT_AND_BACK, GL_LINE);
gl.glBegin(GL2.GL_POLYGON);
gl.glVertex2d(0.0, 0.0);
gl.glVertex2d(50.0, 100.0);
gl.glVertex2d(-50.0, 100.0);
gl.glEnd();
gl.glPolygonMode(GL_FRONT_AND_BACK, GL_FILL);
```



```
gl.glEnable(gl.GL_TEXTURE_2D);
gl.glBindTexture(gl.GL_TEXTURE_2D, texture);
gl.glBegin(GL2.GL_POLYGON);
gl.glTextCoord2d(0.0, 0.0);
gl.glVertex2d(0.0, 0.0);
gl.glTextCoord2d(1.0, 0.0);
gl.glVertex2d(50.0, 0.0);
gl.glTextCoord2d(1.0, 1.0);
gl.glVertex2d(50.0, 50.0);
gl.glTextCoord2d(0.0, 1.0);
gl.glVertex2d(0.0, 50.0);
gl.glEnd();
```



OpenGL

OpenGL 1.1 Vs OpenGL 2.0

"Hello Triangle" in OpenGL 1.1 (5 lines of code)

```
gl.glBegin(GL2.GL_POLYGON);
gl.glVertex2d(0.0, 0.0);
gl.glVertex2d(50.0, 100.0);
gl.glVertex2d(-50.0, 100.0);
gl.glEnd();
```

"Hello Triangle" in OpenGL 2.0 (>100 lines of code)

```
// load our shaders...
try {
    vertexShaderString = Files.readAllLines(Paths.get("./shaders/task3.vert")).toString();
    fragmentShaderString = Files.readAllLines(Paths.get("./shaders/task3.frag")).toString();
} catch (IOException e) {
    // pass
}
matrix = new PMVMatrix();
// setup and load the vertex and fragment shader programs
matrix.glMatrixMode(GL2.GL_PROJECTION);
matrix.glOrthof(0.0f, (float) dim.getWidth(), 0.0f, (float) dim.getHeight(), -1.0f, 1.0f);
matrix.glMatrixMode(GL2.GL_MODELVIEW);
// setup and load the vertex and fragment shader programs
shaderProgram = gl2.glCreateProgram();
vertexShader = gl2.glCreateShader(GL2.GL_VERTEX_SHADER);
String[] vertexShaderArrayStrings = {vertexShaderString};
int[] vertexShaderArrayLengths = {vertexShaderString.length()};
gl2.glShaderSource(vertexShader, 1, vertexShaderArrayStrings, vertexShaderArrayLengths, 0);
... (>80 additonal lines of code...)
```

OpenGL

Texturing

Basic Texturing

- We cover texturing in more detail later on in the course.
- JOGL provides a class to help with texture loading **TextureIO**
- Textures are assigned handles.
- To enable texturing you need to.
  1. enable texture2d with glEnable(GL_TEXTURE2D)
  2. bind the texture you want glBind(...)
  3. define UV coordinates for each vertex using glTexCoord2d(...)
- UV coordinates are always [0, 1].

OpenGL

Common Commands

Common OpenGL Commands

Documentation for the OpenGL API can be found here
https://registry.khronos.org/OpenGL-Refpages/gl2.1/
Some of the more important commands to know are...

- glBegin, glEnd
- glClear, glClearColor
- glPushMatrix, glPopMatrix, glMatrixMode
- glVertex, glTexCoord
- glTranslate, glRotate, glScale

glBegin / glEnd

The **glBegin** and **glEnd** commands are used to delimit the vertices of a **primitive**. Only a limited subset of OpenGL commands with within a begin/end block. Usually we just want to use **glColor**, **glVertex**, and **glTexCoord**.

There are also several primitives to choose from, including

- GL_POINTS
- GL_LINES
- GL_TRIANGLES
- GL_POLYGON (we mostly use this in the labs)

[After the break...]

Shaders...