The Third Dimension!

Game 300

OBJECTIVES

- Explain how Matrices are used in 3D graphics programming.
- Turn a 2D OpenGL Application into 3D.
- Draw primitives to the screen in three dimensions.
- Modify transformation properties of objects within a scene.

TRANSFORMS IN OPENGL 101

- OpenGL uses transformation information to manage things like models, and the area you can see.
- The transformation information managed are the following:
  - Location / position
  - Scale
  - Rotation

- The main transformations in terms of 3D Graphics are:
  - Model: keeps track of the transformation information about the objects within our scene.
    - Could be the crate on the ground, a tree, even our characters.
  - View: the camera position and where we are looking.
  - Projection: This defines the parameters of our scene like how far we can see into depth, up/down, and left to right.
    - Rotation has less of an application here and its typically a combination of position (optionally with scale)
  - Viewport: calculates objects distance in the scene using the above 3 combined.

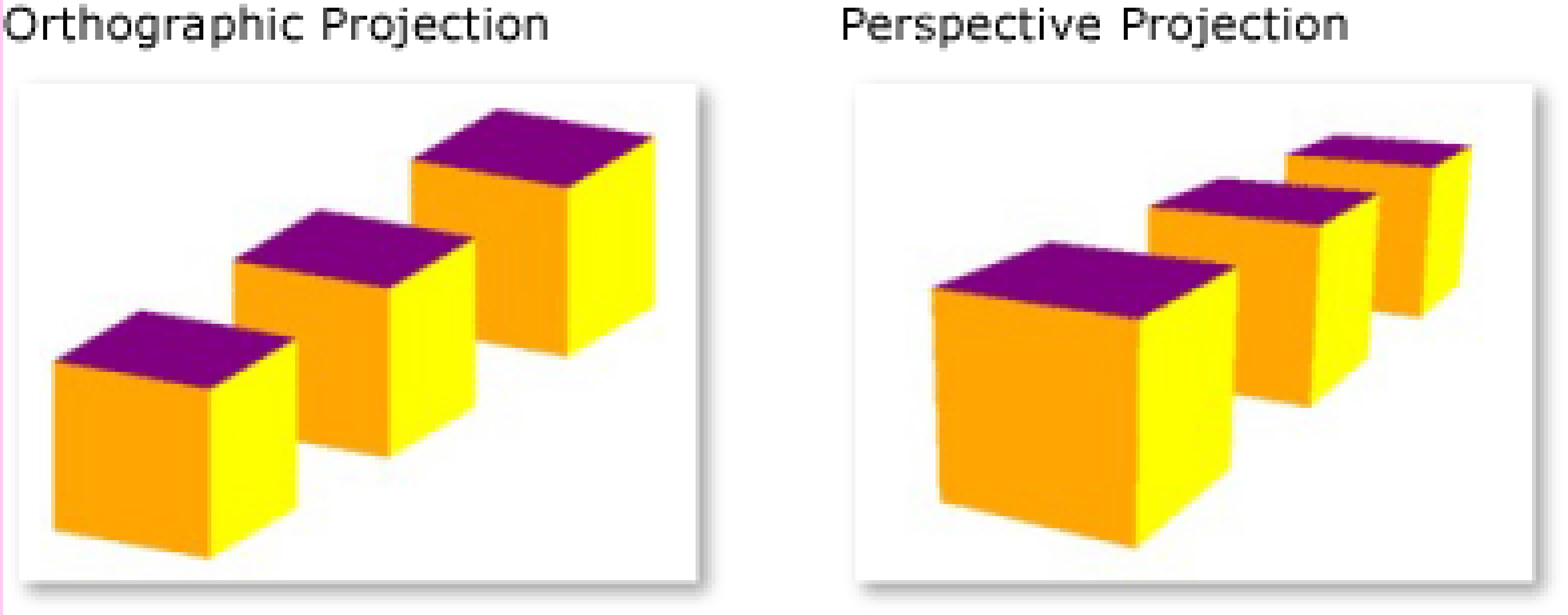
MATRICES IN OPENGL 101

- Matrices in games are used to keep track of a series of modifications to the Transformation properties of objects / views.
- In OpenGL we don’t need to manage all four transformation properties (model, view, projection, viewport)
  - Instead OpenGL has combined matrices to simplify the amount of data floating around.
  - The Types of matrices used:
    - MODELVIEW:
      - This matrix combines the location of our objects (model) in relation to our camera positon (view)
      - Because our camera and our objects exist within the same space, it makes sense for them to share a coordinate system.
    - PROJECTION : used for setting up the dimensions of our scene.
      - This remains it’s own entity as essentially it is defining a rectangular area.
      - If the objects to be rendered do not fit within the area defined they are culled out of the scene.

PERSPECTIVES

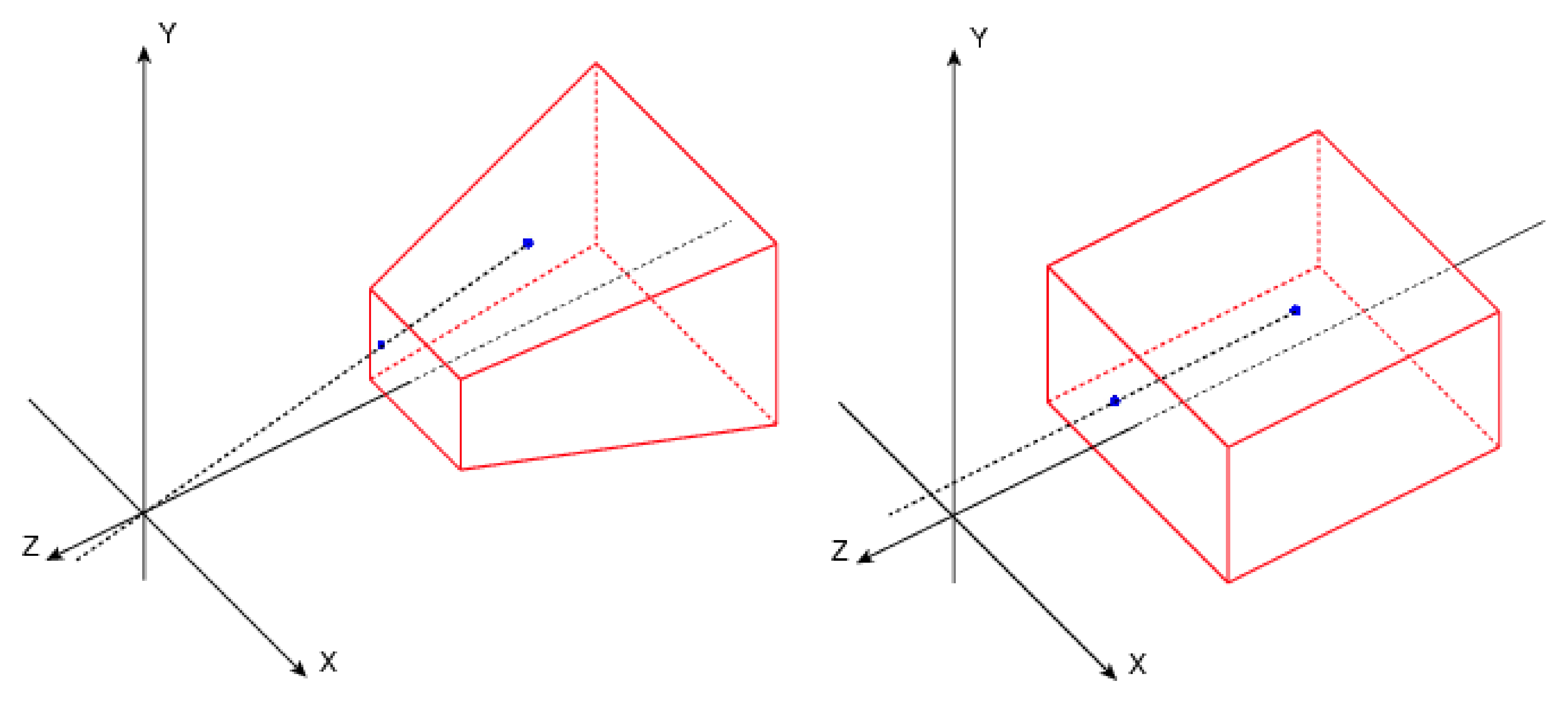
- We will take a look at the VIEW Matrix first.
- There are two main view setups for our camera systems.
- The first view type is orthogonal.
  - This is a somewhat unrealistic view of things and doesn’t have proper perspective with depth.
  - Shapes always have the same size.
  - The viewport in a orthogonal perspective is a cube or box.
- In reality, the further in the distance things are the smaller they appear.
  - This is because of perspective.
  - To achieve perspective, the viewport is a cube where the front face is much smaller than the back face.

PERSPECTIVES



VIEWPORT

- The viewport is the area of the world the camera can see and displays.
- The following is a side by side of what a perspective (left) and a orthographic (right) viewport looks like:



VIEWPORTS IN OPENGL

- In OpenGL we have to strictly specify which Matrix we are dealing with.
  - We can do so by using the glMatrixMode() function:

```
glMatrixMode(GL_PROJECTION);
```

    - The modes available to this call are:
      - GL_MODELVIEW
      - GL_PROJECTION
      - GL_TEXTURE
      - GL_COLOR
- To setup our viewport we will need to use the GL_PROJECTION mode.
- After we set our Matrix mode, we should ensure we are dealing with a blank canvas for our alterations (ensure our matrix is initialized properly)
  - To do this we will reset our currently set Matrix to the **Identity matrix:**

```
glLoadIdentity();
```

IDENTITY MATRIX

- The Identity matrix is the barebones base level of matrices.
- It contains all zeros and a diagonal set of 1's along the entire matrix:
- Any multiplication to the identity matrix just results in the same values.
- It is typically used as the root starting point for all calculations.

$$\begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

HOW ARE MATRICES USED?

- Matrices are used to keep track of a series of manipulations made to geometry in three dimensions.
  - Every time we do an operation like scale, rotate or translate the matrix is modified.
    - Order of these actions are important.
  - The values inside the matrix are automatically modified for us (for now) via the openGL commands.
  - Later we will take a look at the matrix in more detail and how these values are directly altered.

ORTHOGRAPHIC PROJECTION

- We can create an orthographic view (no perspective) which still allows for depth by using the following code:

```
glMatrixMode(GL_PROJECTION);
glLoadIdentity();
glOrtho(-1.0f, 1.0f, -1.0f, 1.0f, 1.0f, 1000.0f);
```

- glOrtho takes in 6 parameters,
  - Far left (-1.0f)
  - Far right (1.0f)
  - Bottom (-1.0f)
  - Top (1.0f)
  - Near (1.0f)
  - Far (1000.0f)
- Near and far typically start and end in a positive value with a larger non-normalized range.

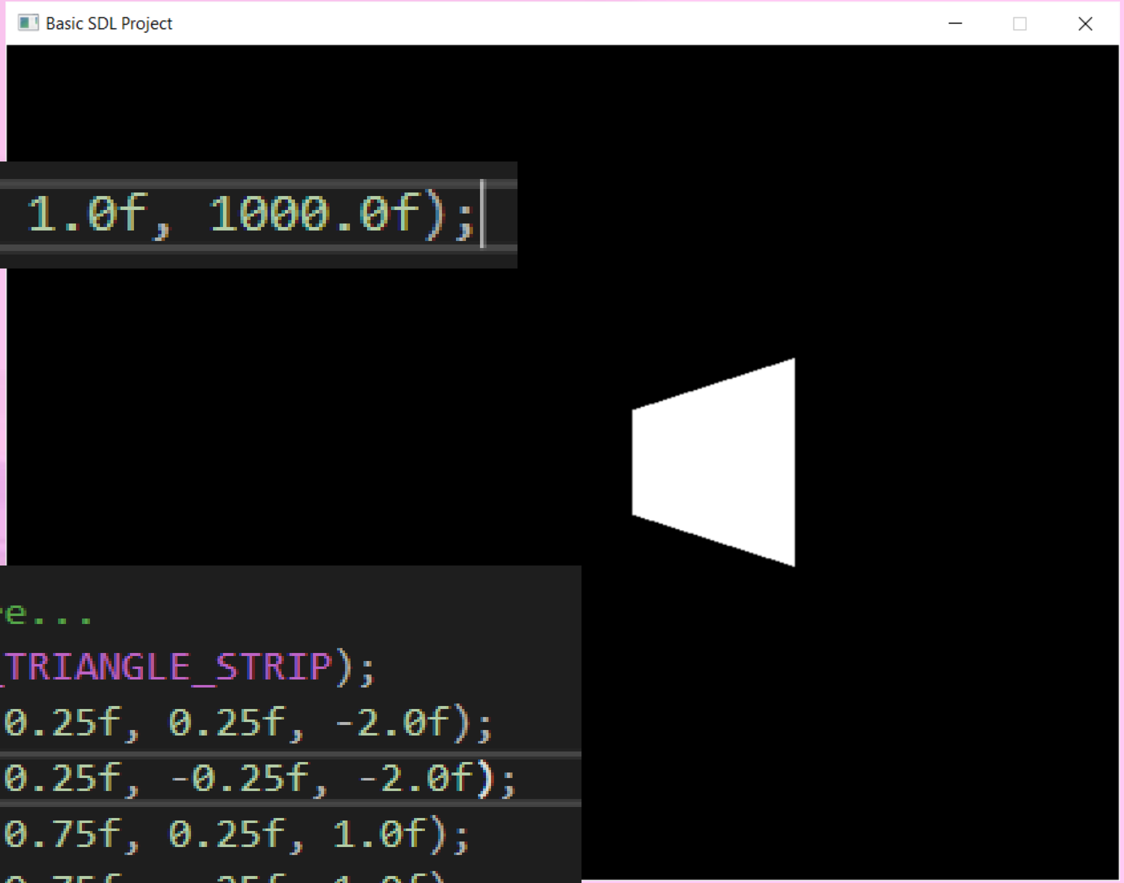
```
glBegin(GL_TRIANGLE_STRIP);
glColor4f(1.0, 0.0f, 0.0f, 1.0f);
glVertex3f(0.25f, 0.25f, -1.0f);
glVertex3f(0.25f, -0.25f, -1.0f);
glVertex3f(0.75f, 0.25f, -5.0f);
glVertex3f(0.75f, -0.25f, -5.0f);
glEnd();
```

PERSPECTIVE PROJECTION

- We can create a perspective view (like the way we see with our eyes), using the frustum function:

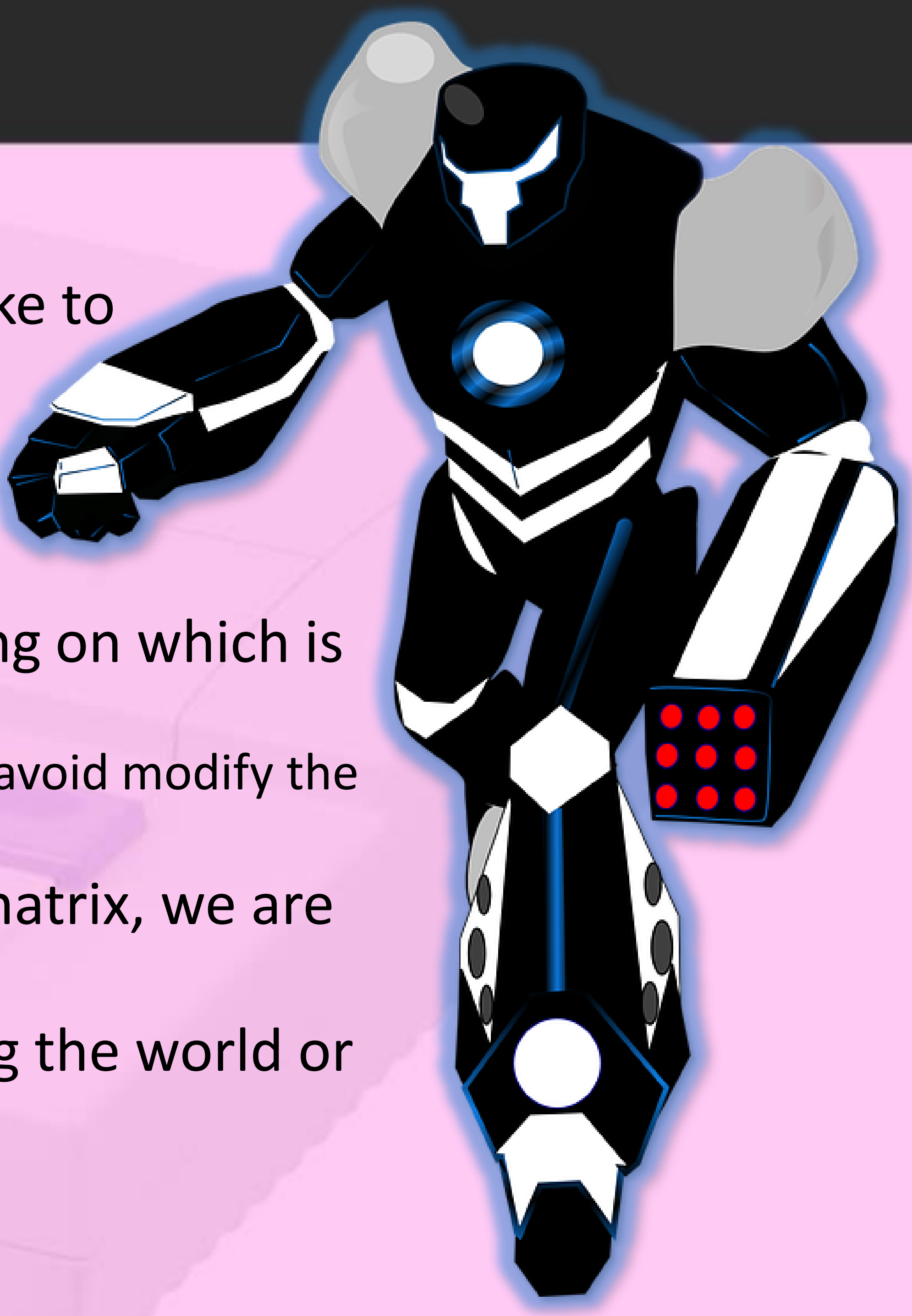
```
glFrustum(-1.0f, 1.0f, -1.0f, 1.0f, 1.0f, 1000.0f);
```

- The frustum function call takes the exact same parameters as the ortho function call.
  - Far left (-1.0f)
  - Far right (1.0f)
  - Bottom (-1.0f)
  - Top (1.0f)
  - Near (1.0f)
  - Far (1000.0f)




```
// draw here...
glBegin(GL_TRIANGLE_STRIP);
glVertex3f(0.25f, 0.25f, -2.0f);
glVertex3f(0.25f, -0.25f, -2.0f);
glVertex3f(0.75f, 0.25f, 1.0f);
glVertex3f(0.75f, -.25f, 1.0f);
glEnd();
```

TRANSFORMATIONS



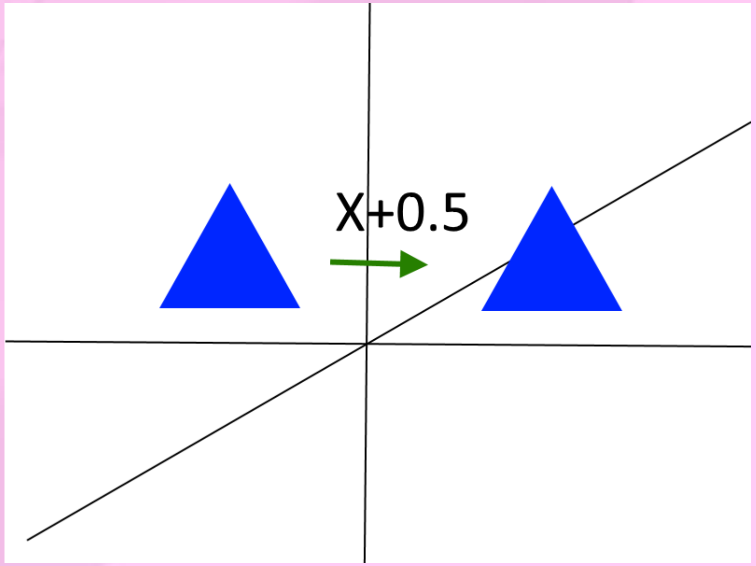
- There are three main modifications we can make to Vector coordinates:
  - Translate
  - Rotate
  - Scale
- We can modify the camera or models depending on which is the active matrix.
  - Be careful when we call our transformation calls to avoid modify the wrong matrix.
- If we have GL_PROJECTION set at the current matrix, we are modifying the camera and viewport.
- If we have GL_MODELVIEW set we are changing the world or object specific matrices.

TRANSLATION

- Translation is the act of moving from one point to another along axis by a floating point value.
- We can translate the current matrix by using the following function:

```
glTranslatef(1.0f, 0.0f, 0.0f);
```

- The glTranslatef function takes in 3 parameters of X, Y, & Z.
  - The values supplied define the amount to move in the plane specified.
- Anything rendered after the translation will be effected

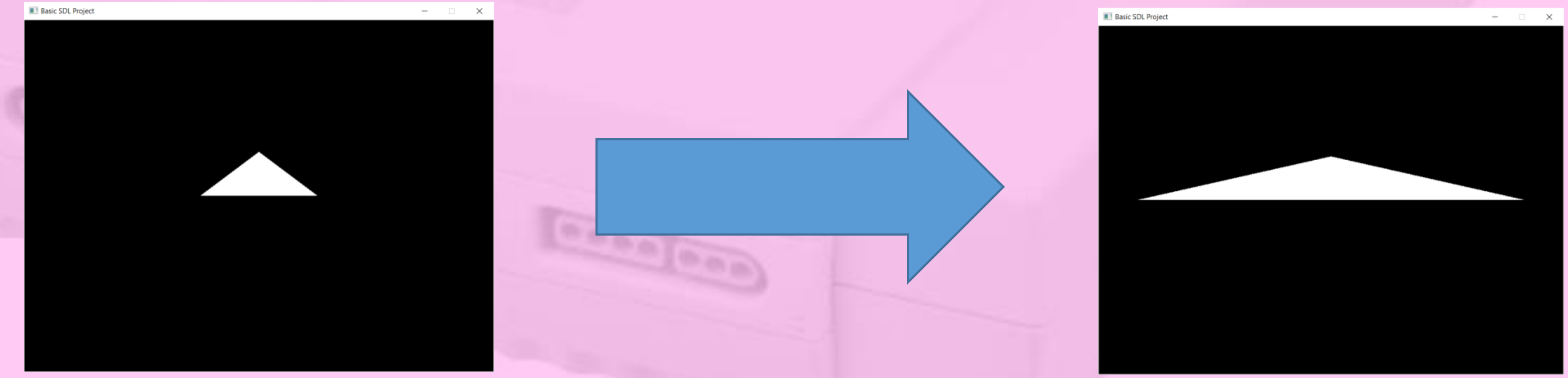


SCALE

- Scaling is the act of increasing or decreasing the size of objects within the matrix.
- We can change the scale using the following function:

```
glScalef(5.0f, 1.5f, 1.5f);
```

- Scaling along an axis scales symmetrically, similar effect to a zoom or stretching effect.
- Similar to translations we control the amount to scale individually per axis.

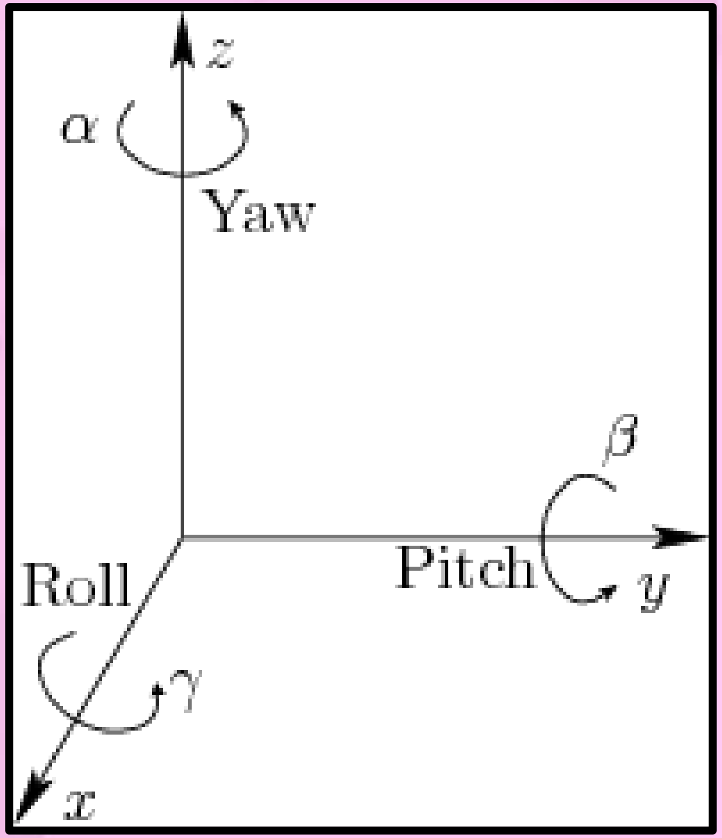


ROTATIONS

- Rotating a matrix will modify the angle of all subsequent draw calls.
- Similar to translations we deal with rotations in three axis.
- When we deal with rotations, we rotate about (or around) an axis.
  - To avoid confusion our rotations have their own terms:
    - Yaw – rotation around the Z axis
    - Pitch – rotation around the y axis
    - Roll – rotations around the x axis
- We can use the following function to add a rotation to the current matrix:

```
glRotatef(45.0f, 0.0f, 1.0f, 0.0f);
```

- The first parameter here is the angle amount to rotate.
- The remaining 3 parameters define the amount to apply the rotation to each axis.



TRANSFORMATION ANIMATIONS

- Using constant values inside of a matrix will result in the same values every frame (assuming you are resetting to the identity every frame).

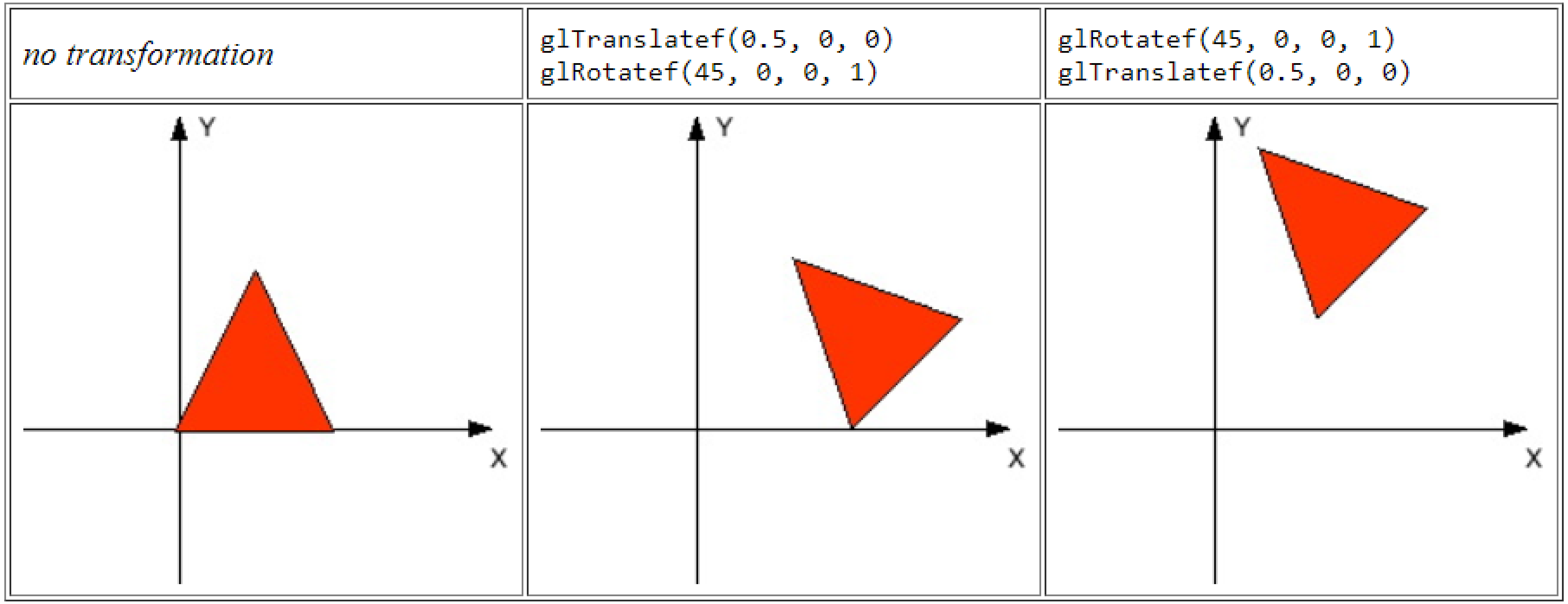
```
glTranslatef(1.0f, 0.0f, 0.0f);
```

- To maintain positons or animate across multiple frames, we should instead use variables inside of our translation amounts.
- In the following example we use a transform struct containing a position Vector containing 3 floats, one for each axis:

```
transform.position.x += 0.01f;
glTranslatef(transform.position.x, transform.position.y, transform.position.z);
```

ORDER OF OPERATIONS

- Its important to note that order of operations for transformation changes are important.
- Rotating around the X axis (roll) then translating will result in much different output than translating first then rotating.



http://resumbrae.com/ub/dms423_f07/05/

SUMMARY

- Explained how Matrices are used in 3D graphics programming.
- Turned a 2D OpenGL Application into 3D.
- Drew primitives to the screen in three dimensions.
- Modified transformation properties of objects within a scene.