Matrices in OpenGL and GLSL

Professor Raymond Zavodnik

Tbilisi State University November 11, 2013

Matrix Use

1. Until Release 3.2 matrices had OpenGL names, e.g. `glScalef()`
2. These matrices have been deprecated in newer releases
3. The user must supply their own matrix package
4. There are packages, e.g. `glm` that can be downloaded
5. In `glm`:
   1. `vec3` and `vec4` represent triples and 4-tupes of e.g. float
   2. `mat3` and `mat4` represent 3 x 3, resp. 4 x 4 matrices of e.g. float
   3. Matrix multiplication is done with overloaded `*`
   4. *All* matrix operations must be executed by the user

Matrix Use

1. Until Release 3.2 matrices had OpenGL names, e.g. `glScalef()`
2. These matrices have been deprecated in newer releases
3. The user must supply their own matrix package
4. There are packages, e.g. `glm` that can be downloaded
5. In `glm`:
    1. `vec3` and `vec4` represent triples and 4-tupes of e.g. float
    2. `mat3` and `mat4` represent 3 x 3, resp. 4 x 4 matrices of e.g. float
    3. Matrix multiplication is done with overloaded `*`
    4. *All* matrix operations must be executed by the user

Matrix Use

1. Until Release 3.2 matrices had OpenGL names, e.g. `glScalef()`
2. These matrices have been deprecated in newer releases
3. The user must supply their own matrix package
4. There are packages, e.g. `glm` that can be downloaded
5. In `glm`:
    1. `vec3` and `vec4` represent triples and 4-tupes of e.g. float
    2. `mat3` and `mat4` represent 3 x 3, resp. 4 x 4 matrices of e.g. float
    3. Matrix multiplication is done with overloaded `*`
    4. *All* matrix operations must be executed by the user

Matrix Use

1. Until Release 3.2 matrices had OpenGL names, e.g. `glScalef()`
2. These matrices have been deprecated in newer releases
3. The user must supply their own matrix package
4. There are packages, e.g. `glm` that can be downloaded
5. In `glm`:
    1. `vec3` and `vec4` represent triples and 4-tupes of e.g. float
    2. `mat3` and `mat4` represent 3 x 3, resp. 4 x 4 matrices of e.g. float
    3. Matrix multiplication is done with overloaded `*`
    4. *All* matrix operations must be executed by the user

Matrix Use

1. Until Release 3.2 matrices had OpenGL names, e.g. `glScalef()`
2. These matrices have been deprecated in newer releases
3. The user must supply their own matrix package
4. There are packages, e.g. `glm` that can be downloaded
5. In `glm:`
   1. `vec3` and `vec4` represent triples and 4-tupes of e.g. float
   2. `mat3` and `mat4` represent 3 x 3, resp. 4 x 4 matrices of e.g. float
   3. Matrix multiplication is done with overloaded `*`
   4. *All* matrix operations must be executed by the user

Translation

- $P(x, y, z) \rightarrow P(x + a, y + b, z + c)$
- Is not a linear transformation, rather *affine*
- Use homogenous coordinates

$$T_{a,bc} = \begin{pmatrix} 1 & 0 & 0 & a \\ 0 & 1 & 0 & b \\ 0 & 0 & 1 & c \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

Rotation About z-Axis

- Main rotations are about the x,y and z-axes
- With these can rotate about an arbitrary axis
- Although this is a linear transformation, we can formulate a matrix version using homogenous coordinates
- Rotating about the z-axis is easy–just use the 2-dimensional formula in the (x, y)-plane (z is constant):

$$R_z(\theta) = \begin{pmatrix} \cos(\theta) & -\sin(\theta) & 0 & 0 \\ \sin(\theta) & \cos(\theta) & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

Rotation About x-Axis

- Measured angle: $y \rightarrow z$
- Just substitute y for x and z for y
- x remains constant

$$R_x(\theta) = \begin{pmatrix} 1 & 0 & 0 & \\ 0 & \cos(\theta) & -\sin(\theta) & 0 \\ 0 & \sin(\theta) & \cos(\theta) & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

Rotation About y-Axis

- This time the measured angle is: $z \rightarrow x$
- Orientation in (x, y, z) space measures from x to z
- Therefore θ is replaced by $-\theta$ (y is constant)

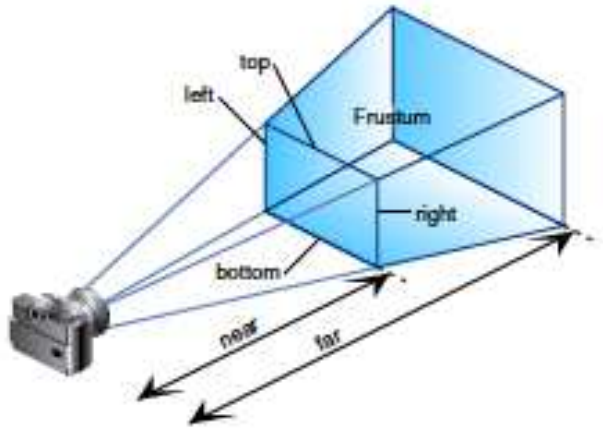
$$R_y(\theta) = \begin{pmatrix} \cos(\theta) & 0 & \sin(\theta) & 0 \\ 0 & 1 & 0 & 0 \\ -\sin(\theta) & 0 & \cos(\theta) & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

Scaling

- Scaling used to stretch/shrink *and* to reflect
- Scaling by zero not recommended
- Scaling makes lighting calculations unwieldy

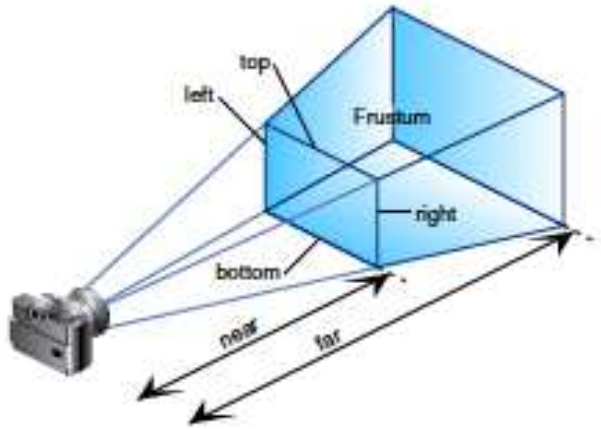
$$S_{a,b,c}(\theta) = \begin{pmatrix} a & 0 & 0 & 0 \\ 0 & b & 0 & 0 \\ 0 & 0 & c & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

Perspective: A Nonlinear Process



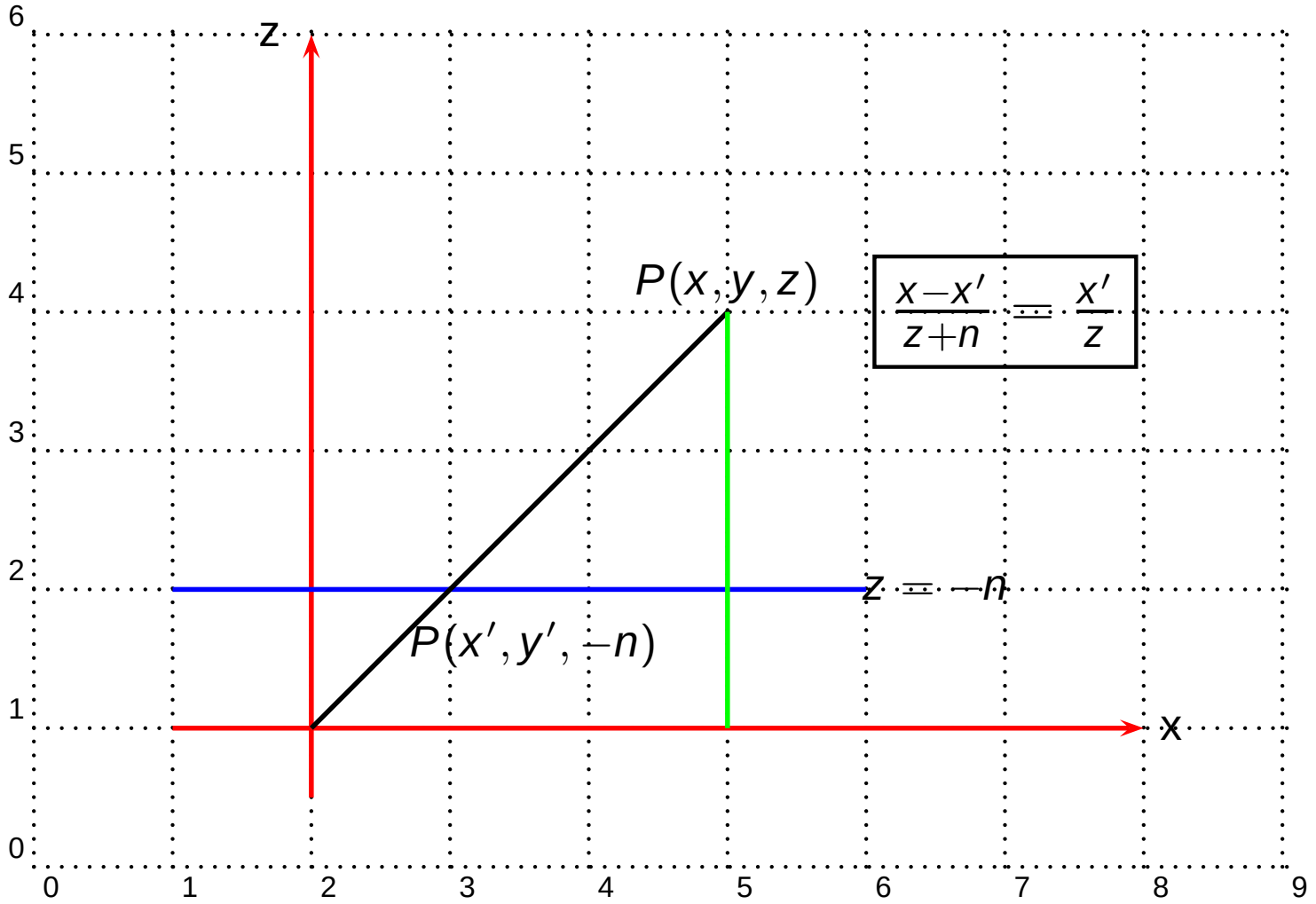
1. Idea: To every point inside the above frustum, or *viewing volume*, the line from the viewer to that point intersects the *viewing plane*, here the near clipping plane, in exactly one point
2. Corresponds to Alberti's dictum of looking out into the world through a (real) window

Perspective: A Nonlinear Process



1. Notice that the frustum is the intersection of 6 clipping planes
2. The right and left sides are given by the planes $x = \pm\alpha z$, top and bottom by $y = \pm\beta z$
3. The front plane is given by $z = near$, the far plane by $z = far$
4. Assume the viewing plane (window) is given by the coordinates $(-l, -b, n)$ and (r, t, n), where l is the left

The Basic Principle



Calculating Perspective

- From $\frac{x-x'}{z+n} = \frac{x'}{z}$ follows $x' = n \times \frac{x}{-z}$
- Similarly $y' = n \times \frac{y}{-z}$
- Note that $-l \leq x' \leq r$ and $-b \leq y' \leq t$
- Now make the following assumption(s): The viewing window is centered at $x = 0$ and $y = 0$, i.e. the window is given by $-r \leq x\prime \leq r$, $-t \leq y\prime \leq t$ and $z' = -n$
- Assume that the observer is at $(0, 0, 0)$ and looking down the negative z-Axis
- If the projection is not *central*, i.e. the above does not hold then the viewing volume must be symmetrized with shearing transformations, likewise given by 4 x 4 matrices, but not discussed here

Calculating Perspective: Step 1

- Scale the frustum so that its sides in x and y make a 45 degree angle with the (x, z) and (y, z) axes
- Thus the sides of the frustum are given by $x = \pm z$ and $y = \pm z$
- There is no scaling in z (yet)
- Since the equations of the bounding planes are $x = \frac{\pm r}{n}$, etc. we must scale in $\frac{n}{r}$ to obtain the proper slopes

Scaling Matrix to Normalize Frustum

$$S_1 = \begin{pmatrix} \frac{n}{r} & 0 & 0 & 0 \\ 0 & \frac{n}{t} & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

Calculating Perspective: Step 2

- Map the resulting frustum to the cube $-1 \leq x, y, z \leq 1$ centered at the origin
- Then project by deleting the z-coordinate
- NDC is given in *floating point*
- We have not yet performed any operations on z
- Use homogeneous coordinates

Matrix Form of Frustum-To-Cube Transformation

$$S_2 = \begin{pmatrix} \frac{n}{r} & 0 & 0 & 0 \\ 0 & \frac{n}{t} & 0 & 0 \\ 0 & 0 & \alpha & \beta \\ 0 & 0 & -1 & 0 \end{pmatrix}$$

for unknown α and β

What????

- Where did that -1 in row 4 and column 3 come from?
- This is where using homogeneous coordinates come in
- Apply the following (simplified) matrix to a point to find out:

$$\begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 1 \\ 0 & 0 & -1 & 0 \end{pmatrix} \begin{pmatrix} x \\ y \\ z \\ 1 \end{pmatrix} = \begin{pmatrix} x \\ y \\ z \\ -z \end{pmatrix}$$

Use Homogeneous Coordinates

- Now divide by the 4-th coordinate to find out to which 3D-point this corresponds
- This maps: $(x, y, z) \rightarrow (\frac{x}{-z}, \frac{y}{-z}, -1)$ *after* dividing by w
- This is called *perspective divide*

Now Determine α and β

- The equation

$$S_2 \cdot \begin{pmatrix} x \\ y \\ z \\ 1 \end{pmatrix} = \begin{pmatrix} \frac{n}{r}x \\ \frac{n}{t}y \\ \alpha z + \beta \\ -z \end{pmatrix}$$

- Must imply that
  1. $z = -n \rightarrow z = +1$ (near plane)
  2. $z = -f \rightarrow z = -1$ (far plane)

Obtain 2 Equations In 2 Unknowns

1. $\frac{\alpha(-n)+\beta}{n} = +1$ (near plane)
2. $\frac{\alpha(-f)+\beta}{f} = -1$ (far plane)

- From which it follows:
  1. $\alpha = +\frac{(f+n)}{f-n}$
  2. $\beta = \frac{2fn}{f-n}$
- Notice the scale in z, i.e. $f - n$

The Final Matrix

$$S_2 = \begin{pmatrix} \frac{n}{r} & 0 & 0 & 0 \\ 0 & \frac{n}{t} & 0 & 0 \\ 0 & 0 & \frac{(f+n)}{f-n} & \frac{2fn}{f-n} \\ 0 & 0 & -1 & 0 \end{pmatrix}$$

- This transformation sends the observer to infinity
- This transformation actually reduced the perspective mapping to orthographic

Polygon Filling Example