CSE 167

DISCUSSION 2

Announcements

- Project 2 is due next Friday
  - Deadline is 02:00PM Friday
  - You may submit to TritonEd as many times as you want before the deadline
  - Grading will be done in B260 and B270
  - Write your name and station number on the board: you will be graded FIFO

Contents

- Some stuff about parsing faces
- Linear algebra needed for this project
- Vertex transformation
    - Matrix multiplication
    - Orbit vs spin
- glm functions
    - Examples
    - Common confusion

Some stuff about parsing faces

- Indices start from 1 not 0!!!
- Indices are stored as unsigned int not glm::vec3!

Linear algebra: homogenous coordinate

- (xz, yz, z) is called a set of homogenous coordinates of (x, y)
  - Note that since z is nonzero, (xz, yz, z) can also be written as (x, y, 1)
- What does this mean geometrically?
  - Chalkboard time
- Why do we need this?
  - Given (x, y, z) we can extend this to a homogenous coordinate (x, y, z, w) with w = 1
  - This means a 3D point can be represented as a 4D vector
  - Then we can multiply 4 X 4 matrices and 4 X 1 vectors
  - So what?

Linear algebra: homogenous coordinate

Let R_y be a rotation matrix with respect to the y-axis and $\theta = 90$:

$$R_y = \begin{bmatrix} cos\theta & 0 & sin\theta \\ 0 & 1 & 0 \\ -sin\theta & 0 & cos\theta \end{bmatrix} = \begin{bmatrix} 0 & 0 & 1 \\ 0 & 1 & 0 \\ -1 & 0 & 0 \end{bmatrix}$$

And let T be a translation vector:

$$T = \begin{bmatrix} 0 \\ 2 \\ 3 \end{bmatrix}$$

To rotate a point $A(-1, 0, 0)$ by 90 degrees and then translate by T:

$$A' = R_y \cdot A + T$$

In other words,

$$A' = \begin{bmatrix} 0 & 0 & 1 \\ 0 & 1 & 0 \\ -1 & 0 & 0 \end{bmatrix} \begin{bmatrix} -1 \\ 0 \\ 0 \end{bmatrix} + \begin{bmatrix} 0 \\ 2 \\ 3 \end{bmatrix} = \begin{bmatrix} 0 \\ 2 \\ 4 \end{bmatrix}$$

So this is a combination of matrix multiplication AND matrix addition.

Linear algebra: homogenous coordinate

But what if you started off with 4×4 matrices and 4×1 vector in the first place?

$$R_y = \begin{bmatrix} cos\theta & 0 & sin\theta & 0 \\ 0 & 1 & 0 & 0 \\ -sin\theta & 0 & cos\theta & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} = \begin{bmatrix} 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 \\ -1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \text{ and } T = \begin{bmatrix} 1 & 0 & 0 & t_x \\ 0 & 1 & 0 & t_y \\ 0 & 0 & 1 & t_z \\ 0 & 0 & 0 & 1 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 2 \\ 0 & 0 & 1 & 3 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

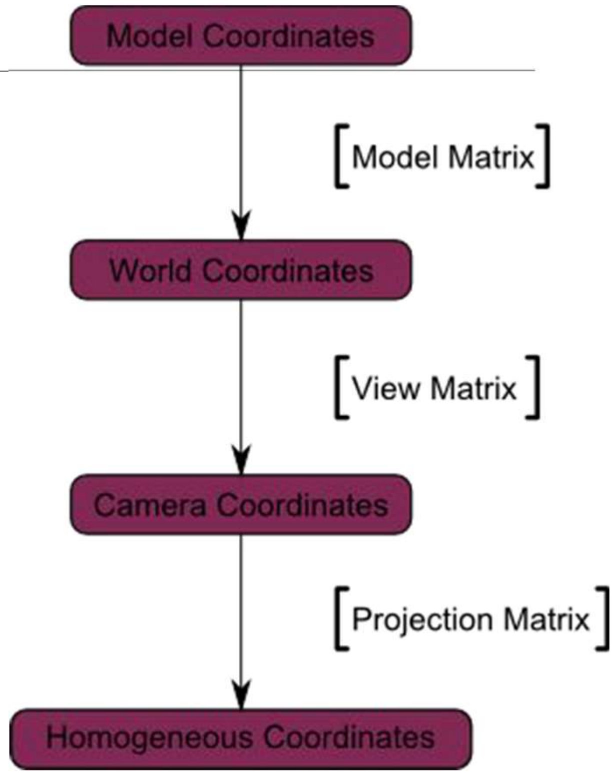
And let $A(-1, 0, 0, 1)$. Then,

$$A' = T \cdot R_y \cdot A$$

In other words, using homogenous coordinate and 4×4 transformation matrices, we can transform a point by just a series of matrix multiplications.

Linear algebra: MVP matrices

- M: place the object
- V: place the camera
- P: set up the camera
- Chalkboard time



http://www.opengl-tutorial.org/beginners-tutorials/tutorial-3-matrices/

Linear algebra: matrix multiplication

- In what order do we multiply?
  - Let's say I have a transformation matrix M that was the result of the previous example
  - If I want to rotate an object with respect to the world's y-axis, which one is right?
    - M = R * M?
    - M = M * R?
  - If I want to rotate an object with respect to its own y-axis again, which one is right?
    - M = R * M?
    - M = M * R?
- If you understood this part, you now know what M is actually the "toWorld" matrix in the starter code

glm functions

- glm::translate()
- glm::rotate()
- glm::scale()

- glm::lookAt()

- glm::perspective()

Vertex transformation: matrix multiplication

Why does the order matter?

- How can you tell the bunny was scaled by its coordinate system or the world coordinate system?
- Example scenario I: scale then translate
- Example scenario II: translate then scale

Vertex transformation: matrix multiplication

```
S = [2 0 0 0   A = [1
     0 1 0 0         0
     0 0 2 0         1
     0 0 0 1]        1]

T = [1 0 0 1   B = [1
     0 1 0 0         0
     0 0 1 1        -1
     0 0 0 1]        1]
```

Vertex transformation: matrix multiplication

Example scenario I: scale then translate

- A’ = S * A = [2 0 2 1]
- B’ = S * B = [2 0 -2 1]
- A” = T * A’ = T * S * A = [3 0 3 1]
- B” = T * B’ = T * S * B = [3 0 -1 1]

Chalkboard time!

Vertex transformation: matrix multiplication

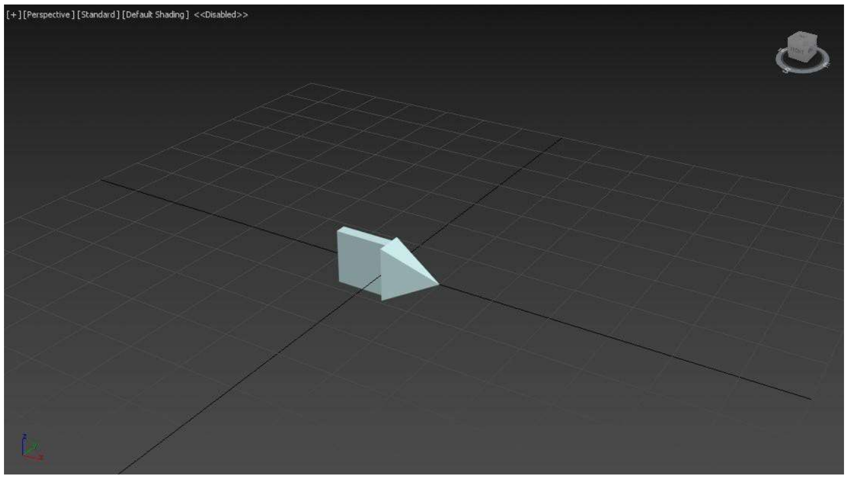
Example scenario II: translate then scale

- A' = T * A = [2 0 2 1]
- B' = T * B = [2 0 0 1]
- A" = S * A' = S * T * A = [4 0 4 1]
- B" = S * B' = S * T * B = [4 0 0 1]

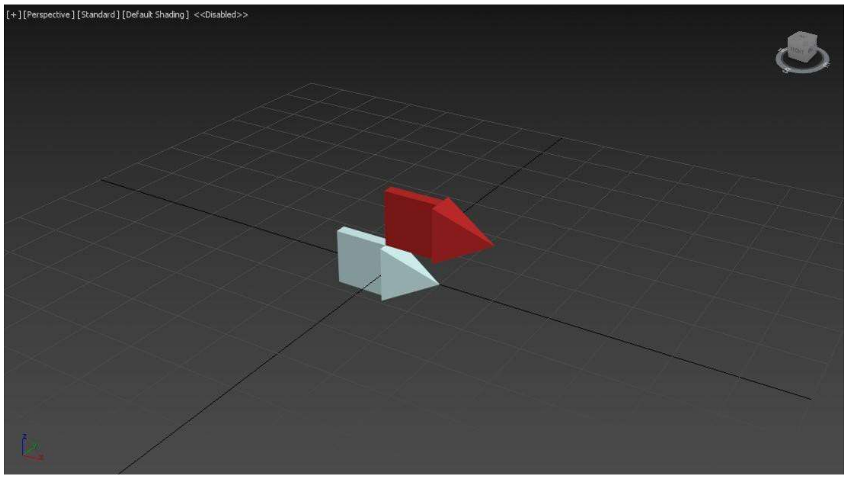
Chalkboard time!

Vertex transformation: spin vs orbit?

Step01

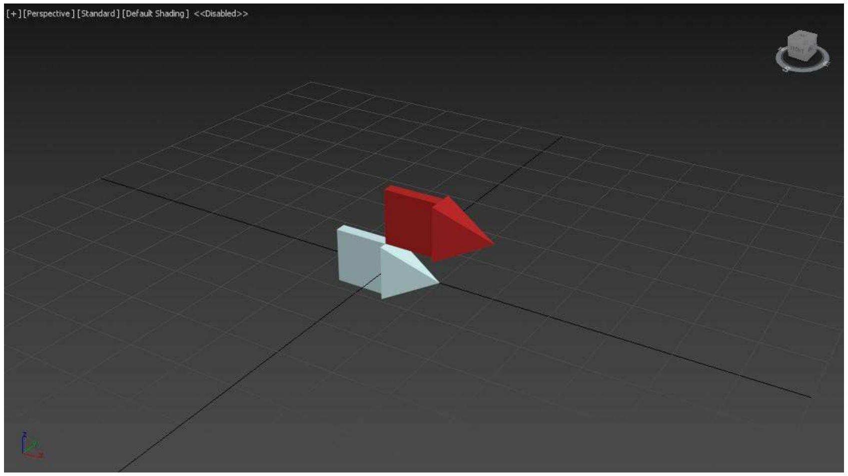


Step02: translation

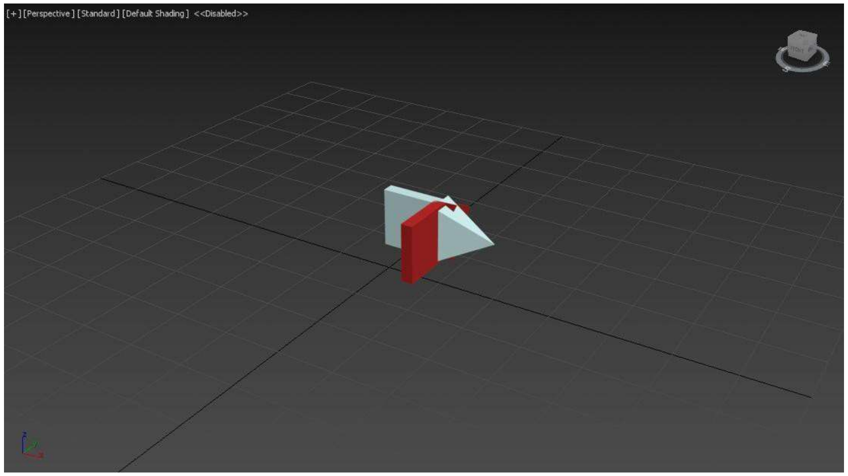


Vertex transformation: spin vs orbit?

Step02: translation

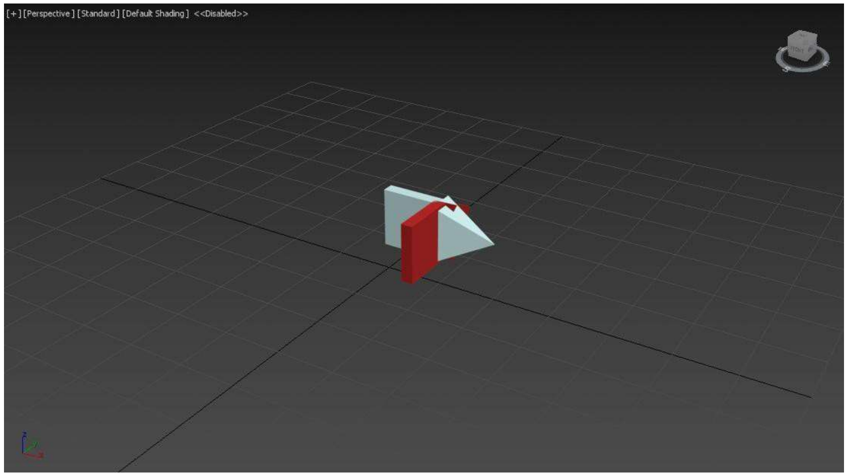


Step03: spin 90 degrees



Vertex transformation: spin vs orbit?

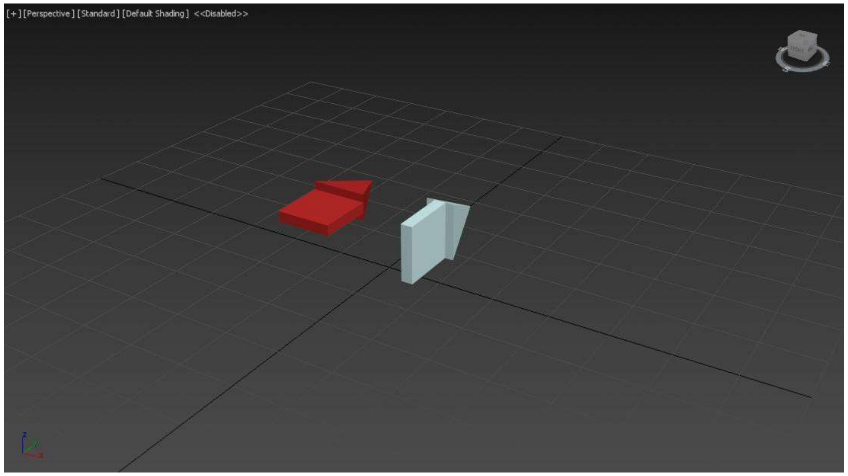
Step03: spin 90 degrees



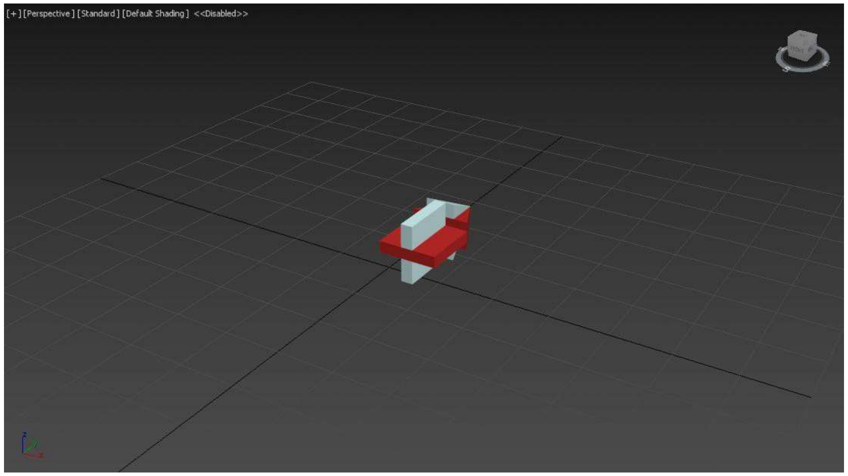
Step04: orbit 90 degrees?

Vertex transformation: spin vs orbit?

Step04: orbit 90 degrees scenario I?

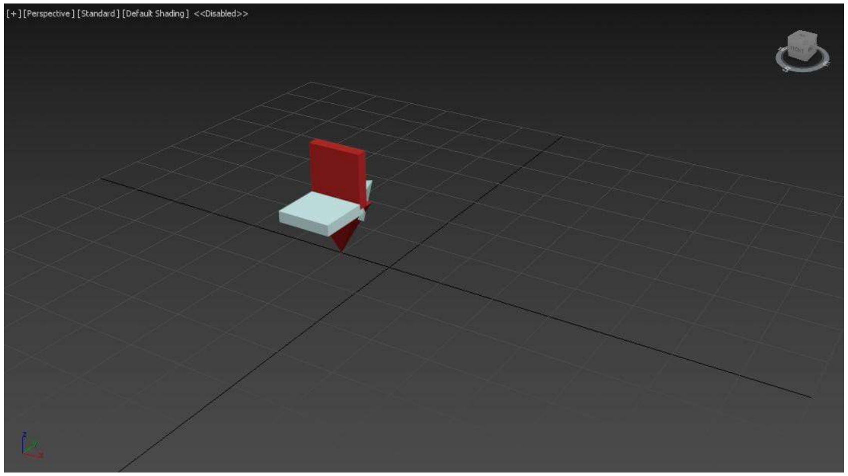


Step04: orbit 90 degrees scenario II?

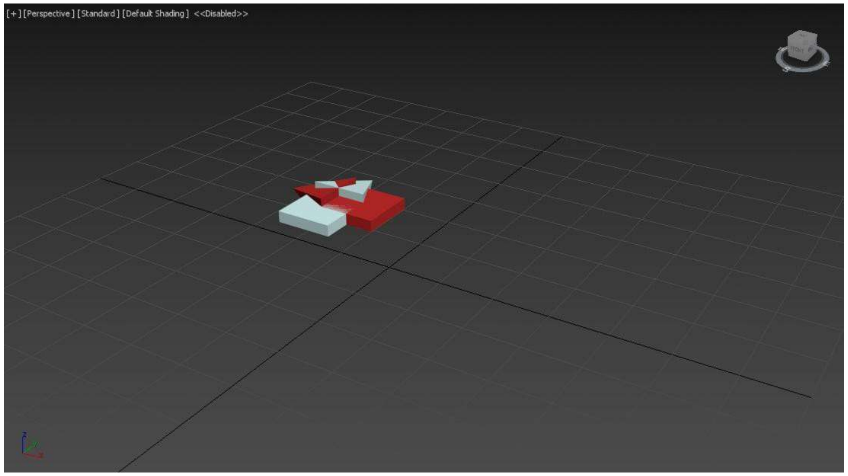


Vertex transformation: spin vs orbit?

Step05: spin 90 degrees scenario I?



Step05: spin 90 degrees scenario II?



Vertex transformation: spin vs orbit?

Step				
01	M1 = I			
02	M2 = T * M1	M2 = M1 * T		
03	M3 = Ry * M2	M3 = M2 * Ry	M3 = Rz * M2	M3 = M2 * Rz
04	M4 = Ry * M3	M4 = M3 * Ry	M4 = Rz * M3	M4 = M3 * Rz
05	M5 = Ry * M4	M5 = M4 * Ry	M5 = Rz * M4	M5 = M4 * Rz

Vertex transformation: spin vs orbit?

Step				
01	M1 = I			
02	M2 = T * M1	M2 = M1 * T		
03	M3 = Ry * M2	M3 = M2 * Ry	M3 = Rz * M2	M3 = M2 * Rz
04	M4 = Ry * M3	M4 = M3 * Ry	M4 = Rz * M3	M4 = M3 * Rz
05	M5 = Ry * M4	M5 = M4 * Ry	M5 = Rz * M4	M5 = M4 * Rz

Again, order matters!

glm::functions: examples

M1 = I	M1 = glm::mat4(1.0f)
M2 = T * M1	M2 = glm::translate(glm::mat4(1.0f), glm::vec3(x, y, z)) * M1
M3 = M2 * Ry	M3 = M2 * glm::rotate(glm::mat4(1.0f), degree, glm::vec3(0, 1, 0))
M4 = Rz * M3	M4 = glm::rotate(glm::mat4(1.0f), degree, glm::vec3(0, 0, 1)) * M3
M5 = M4 * Ry	M5 = M4 * glm::rotate(glm::mat4(1.0f), degree, vec3(0, 1, 0))

glm::functions: common confusion

What are the differences?

- this->toWorld = glm::translate(glm::mat4(1.0f), glm::vec3(x, y, z)) * this->toWorld;
- this->toWorld = this->toWorld * glm::translate(glm::mat4(1.0f), glm::vec3(x, y, z));
- this->toWorld = glm::translate(this->toWorld, glm::vec3(x, y, z));

glm::functions: common confusion

What are the differences given degree = PI / 180 and total_degree = PI / 2?

- this->toWorld = glm::rotate(glm::mat4(1.0f), degree, glm::vec3(0, 1, 0));
- this->toWorld = glm::rotate(glm::mat4(1.0f), degree, glm::vec3(0, 1, 0)) * this->toWorld;
- this->toWorld = this->toWorld * glm::rotate(glm::mat4(1.0f), degree, glm::vec3(0, 1, 0));
- total_degree += degree; this->toWorld = glm::rotate(glm::mat4(1.0f), total_degree, glm::vec3(0, 1, 0));

glm::functions: projection and camera

glm::perspective(FOV, aspect_ratio, near, far)

FOV = how much to view

Aspect_ratio = width/height

Near = nearest boundary

Far = farthest boundary

glm::lookAt(eye, center, up)

Eye: where is the camera

Center: where is the camera looking at

Up: what is the camera's y-axis

You should utilize these functions when writing rasterizer!!!

What values go inside FOV, aspect_ratio, near, far, eye, center, and up?