VGP351 – Week 3

- Agenda:
  - Quiz #1
  - Transformations
    - Modeling
    - Viewing
    - Projection

© Copyright Ian D. Romanick 2009 - 2011

Coordinate Spaces

- Is the spaceship moving, or is the viewer moving?

© Copyright Ian D. Romanick 2009 - 2011

Coordinate Spaces

- Relativistically, it *doesn't matter*
  - Pick the reference frame that's most convenient at the time

© Copyright Ian D. Romanick 2009 - 2011

Coordinate Spaces

- Coordinates are always relative to some “space”
  - Object space: Local coordinate system of the object
  - World space: Global coordinate system relative to the 3D “world”
  - Eye / camera space: Coordinate system relative to the viewer
- When we translate objects relative to other objects, we may talk about other spaces
  - If the hand of a 3D model is rotated relative to the arm of the model, we may talk about “hand-space” or “arm-space”

© Copyright Ian D. Romanick 2009 - 2011

Coordinate Spaces

- Watch your coordinate spaces!
  - When performing calculations, be sure that the coordinate spaces match
    - Measuring distances between points
    - Measuring angles between vectors
    - Performing transformations
  - Just like being careful of units in physics / chemistry equations
    - If an acceleration calculation comes out in Newtons (kg m/s^2) instead of m/s^2, you *know* there's an error

© Copyright Ian D. Romanick 2009 - 2011

Coordinate Spaces

- Variable names should convey the coordinate space

```
vec3 normal_ws;     // normal in world-space
vec4 position_es;   // position in eye-space
vec4 light_ws;      // light pos in world-space
vec3 light_dir_ss;  // light direction in surface-
                    // space

// Obviously wrong!
light_dir_ss = (light_ws – position_es).xyz;

// Not obvious, but still wrong
light_dir = (light – position).xyz;
```


© Copyright Ian D. Romanick 2009 - 2011

Orthonormal Basis

- It's a mouthful...what does it mean?
- A vector space where all of the components are *orthogonal* to each other, and each is *normal*
  - Normal meaning unit length
  - Orthogonal meaning at right angles
    - The *other* meaning of normal
- Every pure rotation matrix (i.e., no scaling) is an orthonormal basis
  - As is the identity matrix

© Copyright Ian D. Romanick 2009 - 2011

Viewing

- Q: Given a world position for a camera, a world position to point the camera at, and an “up” direction, how can we construct a transformation using just rotations and translations?

© Copyright Ian D. Romanick 2009 - 2011

Viewing

- Q: Given a world position for a camera, a world position to point the camera at, and an “up” direction, how can we construct a transformation using just rotations and translations?
- A: We can't. We need 3 vectors to construct an orthonormal basis
  - [Hughes 99] presents a method to construct from just one vector, but it has limitations

© Copyright Ian D. Romanick 2009 - 2011

Viewing

- Given:
  - $\mathbf{e}$: Position of the eye (or camera) in world-space
  - $\mathbf{v}$: The point being viewed
  - $\mathbf{u}$: the “up” direction
- Calculate the unit vector from the viewpoint to the eye:

$$\mathbf{f}' = \mathbf{v} - \mathbf{e}$$

$$\mathbf{f} = \frac{\mathbf{f}'}{|\mathbf{f}'|}$$

  - This is the Z axis

© Copyright Ian D. Romanick 2009 - 2011

Viewing

- Calculate a vector orthogonal to the Z-axis and the up vector:

$$\mathbf{s} = \mathbf{f} \times \mathbf{u}$$

  - This is the X-axis

© Copyright Ian D. Romanick 2009 - 2011

Viewing

- Calculate a vector orthogonal to the Z-axis and the up vector:

$$\mathbf{s} = \mathbf{f} \times \mathbf{u}$$

  - This is the X-axis
- Calculate a vector orthogonal to the X-axis and the Z-axis:

$$\mathbf{t} = \mathbf{s} \times \mathbf{f}$$

  - This is the Y-axis
  - Why can't we just use $\mathbf{u}$?

© Copyright Ian D. Romanick 2009 - 2011

Viewing

- Drop these vectors into a matrix:

$$\mathbf{M}_\mathrm{v} = \begin{bmatrix} \mathbf{s}_0 & \mathbf{s}_1 & \mathbf{s}_2 & 0 \\ \mathbf{t}_0 & \mathbf{t}_1 & \mathbf{t}_2 & 0 \\ -\mathbf{f}_0 & -\mathbf{f}_1 & -\mathbf{f}_2 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \times \begin{bmatrix} 1 & 0 & 0 & -\mathbf{e}_0 \\ 0 & 1 & 0 & -\mathbf{e}_1 \\ 0 & 0 & 1 & -\mathbf{e}_2 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

  - The translation moves the eye to the origin

© Copyright Ian D. Romanick 2009 - 2011

References

General information about rotation matrices and orthonormal bases:

http://en.wikipedia.org/wiki/Rotation_matrix

http://www.wikipedia.org/Orthonormal_basis

Really good explanation of arbitrary rotation matrices:

http://www.euclideanspace.com/maths/geometry/rotations/conversions/angleToMatrix/index.htm

Hughes, J. F., and Möller, T. Building an Orthonormal Basis from a Unit Vector. *Journal of Graphics Tools* 4, 4 (1999), 33-35. http://www.cs.brown.edu/research/pubs/authors/john_f._hughes.html

© Copyright Ian D. Romanick 2009 - 2011

Projection

- Once objects are transformed to camera-space, they're still 3D
  - The screen is still 2D
  - Camera parameters (e.g., field of view) need to be applied
- Four steps remain:
  - Projection from camera space to clip coordinates
    - A cube on the range ±1
  - Perspective divide
  - Map clip coords to normalized device coords (NDC)
    - X and Y in ±1, Z in [0,1]
  - Map NDC to pixel coordinates

© Copyright Ian D. Romanick 2009 - 2011

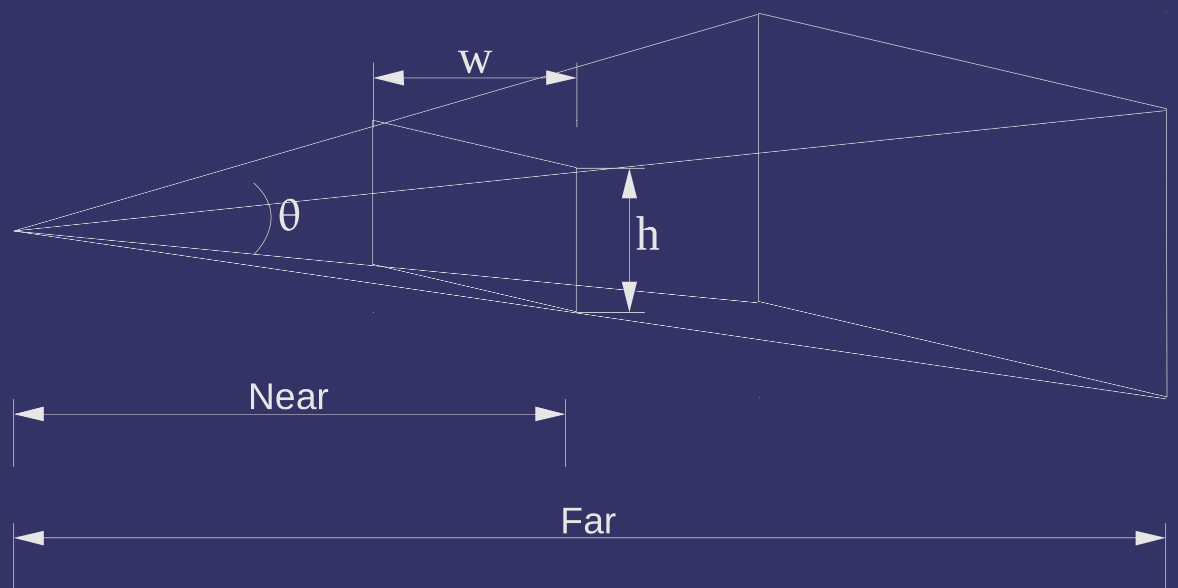
Projection

- Perspective:
  - Simulates visual foreshortening caused by the way light projects onto the back of the eye
  - Represents the view volume with a frustum (a pyramid with the top cut off)
  - The real work is done by dividing X and Y by Z
- Orthographic:
  - Represents the view volume with a cube
  - Also called *parallel projection* because lines that are parallel before the projection remain parallel after

© Copyright Ian D. Romanick 2009 - 2011

Perspective Projection

- A few parameters control the view volume:
  - Near: Distance from the camera to the near viewing plane. Objects in front of this plane will be clipped
  - Far: Distance from the camera to the far viewing plane. Objects behind this plane will be clipped
  - θ: Field-of-view in the Y direction
  - Aspect ratio: Ratio of the width of the view to the height of the view



© Copyright Ian D. Romanick 2009 - 2011

Perspective Projection

$$f = cotan\left(\frac{\theta}{2}\right)$$

$$\mathbf{M}_p = \begin{bmatrix} \frac{f}{aspect} & 0 & 0 & 0 \\ 0 & f & 0 & 0 \\ 0 & 0 & -\frac{far+near}{far-near} & -\frac{2\times far\times near}{far-near} \\ 0 & 0 & -1 & 0 \end{bmatrix}$$

- Limited form of projection matrix that assumes symmetry in X and Y directions
- *near* and *far* are distances
  - We're actually looking down the negative Z axis in camera space

© Copyright Ian D. Romanick 2009 - 2011

Perspective Projection

- **WARNING:**
  - `near` and `far` are reserved words in MS compilers
    - Nice of them to follow the rules of the C specification
    - Dates back to quirks of the old 8086 and 80286 CPUs
  - Maybe use:
    - `hither` and `yon`
    - `zNear` and `zFar`

© Copyright Ian D. Romanick 2009 - 2011

Putting it all together

- Typically have a *modeling* transform, a *viewing* transform, and a *projection*
  - Combine these into a single "model-view-projection" matrix: $\mathbf{M}_{mvp} = \mathbf{M}_p \times \mathbf{M}_v \times \mathbf{M}_m$
  - Transform a vertex by this single matrix:

```
uniform mat4 mvp;
void main(void)
{
        gl_Position = mvp * gl_Vertex;
}
```


© Copyright Ian D. Romanick 2009 - 2011

References

http://en.wikipedia.org/wiki/3D_projection

- Especially the third step: perspective transform

http://en.wikipedia.org/wiki/Orthographic_projection_%28geometry%29

http://en.wikipedia.org/wiki/Isometric_projection

© Copyright Ian D. Romanick 2009 - 2011

Next week...

- Hidden surface removal / occlusion
  - Backface culling
  - Painters algorithm
  - Z-buffer
  - Frustum culling
- Assignment #2, part 1

© Copyright Ian D. Romanick 2009 - 2011

Legal Statement

This work represents the view of the authors and does not necessarily represent the view of Intel or the Art Institute of Portland.

OpenGL is a trademark of Silicon Graphics, Inc. in the United States, other countries, or both.

Khronos and OpenGL ES are trademarks of the Khronos Group.

Other company, product, and service names may be trademarks or service marks of others.

© Copyright Ian D. Romanick 2009 - 2011