Project 1E: questions or concerns?
Project 1F: now due May 8\textsuperscript{th}
Project 2A: now due May 15\textsuperscript{th}
Project 2B: now due May 20\textsuperscript{th}
Final project proposal: now due May 22\textsuperscript{nd}
Our goal

World space:
Triangles in native Cartesian coordinates
Camera located anywhere

Camera space:
Camera located at origin, looking down -Z
Triangle coordinates relative to camera frame

Discuss this today

Image space:
All viewable objects within -1 <= x,y,z <= +1

Screen space:
All viewable objects within -1 <= x, y <= +1

Device space:
All viewable objects within 0<=x<=width, 0 <=y<=height

Discussed this last time, review today
How do we specify a camera?

The “viewing pyramid” or “view frustum”.

Frustum: In geometry, a frustum (plural: frusta or frustums) is the portion of a solid (normally a cone or pyramid) that lies between two parallel planes cutting it.

class Camera
{
    public:
        double near, far;
        double angle;
        double position[3];
        double focus[3];
        double up[3];
};
New terms

- **Coordinate system:**
  - A system that uses coordinates to establish position

- **Example:** (3, 4, 6) really means...
  - $3 \times (1,0,0)$
  - $4 \times (0,1,0)$
  - $6 \times (0,0,1)$

- **Since we assume the Cartesian coordinate system**
New terms

- **Frame**: A way to place a coordinate system into a specific location in a space

- **Cartesian example**: (3,4,6)
  - It is assumed that we are speaking in reference to the origin (location (0,0,0)).

- A frame \( F \) is a collection \( (v_1, v_2, \ldots, v_n, O) \) is a frame over a space if \( (v_1, v_2, \ldots, v_n) \) form a basis over that space.
Example of Frames

- Frame $F = (v_1, v_2, O)$
  - $v_1 = (0, -1)$
  - $v_2 = (1, 0)$
  - $O = (3, 4)$

- What are $F$’s coordinates for the point $(6, 6)$?

- Answer: $(-2, 3)$
Reminder: Cartesian Frame

- World space uses Cartesian Frame
  - \( V_1 = (1,0,0) \)
  - \( V_2 = (0,1,0) \)
  - \( V_3 = (0,0,1) \)
  - \( O = (0,0,0) \)
  - \( P = (x,y,z) = x*V_1+y*V_2+z*V_3 + O \)
Our goal

World space:
- Triangles in native Cartesian coordinates
- Camera located anywhere

Camera space:
- Camera located at origin, looking down -Z
- Triangle coordinates relative to camera frame

- Need to construct a Camera Frame
- Need to construct a matrix to transform points from Cartesian Frame to Camera Frame
  - Transform triangle by transforming its three vertices
Camera frame construction

- Must choose \((v_1, v_2, v_3, O)\)

  ![Diagram showing camera frame construction]

  ```
  class Camera {
    public:
      double near, far;
      double angle;
      double position[3];
      double focus[3];
      double up[3];
  }
  ```

- \(O = \text{camera position}\)

- \(v_3 = O\)-focus
  - Not “focus-\(O\)”, since we want to look down -\(Z\)
What is the up axis?

- Up axis is the direction from the base of your nose to your forehead
What is the up axis?

- Up axis is the direction from the base of your nose to your forehead.
- (If you lie down while watching TV, the screen is sideways.)
Camera frame construction

- Must choose \((v1,v2,v3,O)\)
- \(O = \) camera position
- \(v3 = O\)-focus
- \(v2 = \) up
- \(v1 = \) up \(\times (O\)-focus\)

Camera space:
- Camera located at origin, looking down -Z
- Triangle coordinates relative to camera frame

```c++
class Camera {
    public:
        double near, far;
        double angle;
        double position[3];
        double focus[3];
        double up[3];
};
```
But wait ... what if $\text{dot}(v_2, v_3) \neq 0$?

We can get around this with two cross products:
- $v_1 = \text{Up} \times (O\text{-focus})$
- $v_2 = (O\text{-focus}) \times v_1$
Camera frame summarized

- \( O = \) camera position
- \( v_1 = \text{Up} \times (O\text{-focus}) \)
- \( v_2 = (O\text{-focus}) \times v_1 \)
- \( v_3 = O\text{-focus} \)

```cpp
class Camera {
    public:
        double near, far;
        double angle;
        double position[3];
        double focus[3];
        double up[3];
};
```
What is the cross product?

- $\mathbf{A} \times \mathbf{B} = (A.y \times B.z - A.z \times B.y,$
  $B.x \times A.z - A.x \times B.z,$
  $A.x \times B.y - A.y \times B.x)$

What is the physical interpretation of a cross product?
- Finds a vector perpendicular to both $\mathbf{A}$ and $\mathbf{B}$.
Camera frame: example

- \( O = \) camera position
- \( v_1 = \) Up \( \times (O\text{-focus}) \)
- \( v_2 = (O\text{-focus}) \times v_1 \)
- \( v_3 = O\text{-focus} \)

```cpp
class Camera {
public:
    double  near, far;
    double  angle;
    double  position[3]; = {5,5,5};
    double  focus[3]; = {1,2,4};
    double  up[3]; = {0,1,1};
};
```
Camera frame: example

- \( O = \) camera position = \{ 5, 5, 5 \}
- \( v_1 = \text{Up} \times (O-\text{focus}) = \{0, 1, 1\} \times \{4, 3, 1\} = \{-2, 4, -4\}/\sqrt{36} \)
- \( v_2 = (O-\text{focus}) \times v_1 = \{4, 3, 1\} \times \{-2, 4, -4\} = \{-16, 14, 22\}/\sqrt{936} \)
- \( v_3 = O-\text{focus} = \{4, 3, 1\}/\sqrt{26} \)

```cpp
class Camera {
public:
    double near, far;
    double angle;
    double position[3]; = \{5, 5, 5\};
    double focus[3]; = \{1, 2, 4\};
    double up[3]; = \{0, 1, 1\};
};
```
Our goal

World space:
- Triangles in native Cartesian coordinates
- Camera located anywhere

Camera space:
- Camera located at origin, looking down -Z
- Triangle coordinates relative to camera frame

- Need to construct a Camera Frame
- Need to construct a matrix to transform points from Cartesian Frame to Camera Frame
  - Transform triangle by transforming its three vertices
The Camera Transform

- Our two frames:
  - Cartesian:
    - $<1,0,0>$
    - $<0,1,0>$
    - $<0,0,1>$
    - $(0,0,0)$
  - Camera:
    - $v1 = \text{up} \times (O\text{-focus})$
    - $v2 = (O\text{-focus}) \times u$
    - $v3 = (O\text{-focus})$
    - $O$
The Camera Transform

- Our two frames:
  - Cartesian:
    - <1,0,0>
    - <0,1,0>
    - <0,0,1>
    - (0,0,0)
  - Camera:
    - v1 = up x (O-focus)
    - v2 = (O-focus) x u
    - v3 = (O-focus)
    - O

Camera is a Frame, so we can express any vector in Cartesian as a combination of v1, v2, and v3.
The Camera Transform

- The Cartesian vector \( <1,0,0> \) can be represented as some combination of the Camera basis functions \( v_1, v_2, v_3 \):
  - \( <1,0,0> = e_{1,1} \cdot v_1 + e_{1,2} \cdot v_2 + e_{1,3} \cdot v_3 \)
- So can the Cartesian vector \( <0,1,0> \):
  - \( <0,1,0> = e_{2,1} \cdot v_1 + e_{2,2} \cdot v_2 + e_{2,3} \cdot v_3 \)
- So can the Cartesian vector \( <0,0,1> \):
  - \( <0,0,1> = e_{3,1} \cdot v_1 + e_{3,2} \cdot v_2 + e_{3,3} \cdot v_3 \)
- So can the vector: Cartesian origin – Camera origin:
  - \( (0,0,0) - O = e_{4,1} \cdot v_1 + e_{4,2} \cdot v_2 + e_{4,3} \cdot v_3 \rightarrow \)
  - \( (0,0,0) = e_{4,1} \cdot v_1 + e_{4,2} \cdot v_2 + e_{4,3} \cdot v_3 + O \)
The Camera Transform

- $<1,0,0> = e_{1,1} \cdot v_1 + e_{1,2} \cdot v_2 + e_{1,3} \cdot v_3$
- $<0,1,0> = e_{2,1} \cdot v_1 + e_{2,2} \cdot v_2 + e_{2,3} \cdot v_3$
- $<0,0,1> = e_{3,1} \cdot v_1 + e_{3,2} \cdot v_2 + e_{3,3} \cdot v_3$
- $(0,0,0) = e_{4,1} \cdot v_1 + e_{4,2} \cdot v_2 + e_{4,3} \cdot v_3 + O$

$\rightarrow$

- $[<1,0,0>] = \begin{bmatrix} e_{1,1} & e_{1,2} & e_{1,3} & 0 \end{bmatrix} [v_1]$
- $[<0,1,0>] = \begin{bmatrix} e_{2,1} & e_{2,2} & e_{2,3} & 0 \end{bmatrix} [v_2]$
- $[<0,0,1>] = \begin{bmatrix} e_{3,1} & e_{3,2} & e_{3,3} & 0 \end{bmatrix} [v_3]$
- $(0,0,0) = \begin{bmatrix} e_{4,1} & e_{4,2} & e_{4,3} & 1 \end{bmatrix} [O]$
Consider the meaning of Cartesian coordinates \((x, y, z)\):

\[
\begin{bmatrix}
  x & y & z & 1
\end{bmatrix}
\begin{pmatrix}
  <1, 0, 0> \\
  <0, 1, 0> \\
  <0, 0, 1> \\
  (0, 0, 0)
\end{pmatrix}
= (x, y, z)
\]

But:

\[
\begin{pmatrix}
  <1, 0, 0> \\
  <0, 1, 0> \\
  <0, 0, 1>
\end{pmatrix}
= \begin{pmatrix}
  e_{1,1} & e_{1,2} & e_{1,3} & 0 \\
  e_{2,1} & e_{2,2} & e_{2,3} & 0 \\
  e_{3,1} & e_{3,2} & e_{3,3} & 0
\end{pmatrix}
\begin{pmatrix}
  v_1 \\
  v_2 \\
  v_3
\end{pmatrix}
\]

\[(0, 0, 0) = \begin{pmatrix}
  e_{4,1} & e_{4,2} & e_{4,3} & 1
\end{pmatrix}
\begin{pmatrix}
  O
\end{pmatrix}
\]
The Camera Transform

But:

\[
\begin{bmatrix}
1 & 0 & 0 & 0 \\
x & y & z & 1 \\
0 & 0 & 1 & 0 \\
0 & 0 & 0 & 1
\end{bmatrix}
\]

Coordinates of \((x,y,z)\) with respect to Cartesian frame.

Coordinates of \((x,y,z)\) with respect to Camera frame.

So this matrix is the camera transform!!
## Solving the Camera Transform

\[
\begin{bmatrix}
  e_{1,1} & e_{1,2} & e_{1,3} & 0 \\
  e_{2,1} & e_{2,2} & e_{2,3} & 0 \\
  e_{3,1} & e_{3,2} & e_{3,3} & 0 \\
  e_{4,1} & e_{4,2} & e_{4,3} & 1 \\
\end{bmatrix}
= \begin{bmatrix}
  v_{1,x} & v_{2,x} & v_{3,x} & 0 \\
  v_{1,y} & v_{2,y} & v_{3,y} & 0 \\
  v_{1,z} & v_{2,z} & v_{3,z} & 0 \\
  v_{1,t} & v_{2,t} & v_{3,t} & 1 \\
\end{bmatrix}
\]

Where \( t = (0,0,0)-O \)

How do we know?: Cramer’s Rule + simplifications

Want to derive?:

Let's do an example

\[
\begin{bmatrix}
e_{1,1} & e_{1,2} & e_{1,3} & 0 \\
e_{2,1} & e_{2,2} & e_{2,3} & 0 \\
e_{3,1} & e_{3,2} & e_{3,3} & 0 \\
e_{4,1} & e_{4,2} & e_{4,3} & 1 \\
\end{bmatrix}
= \begin{bmatrix}
v_{1.x} & v_{2.x} & v_{3.x} & 0 \\
v_{1.y} & v_{2.y} & v_{3.y} & 0 \\
v_{1.z} & v_{2.z} & v_{3.z} & 0 \\
v_{1.t} & v_{2.t} & v_{3.t} & 1 \\
\end{bmatrix}
\]

Where \( t = (0,0,0) - O \)

Camera frame:

\( V1 = (0.6, 0, 0.8) \)
\( V2 = (0, 1, 0) \)
\( V3 = (-0.8, 0, 0.6) \)
\( O = (2,2,2) \)
### Let's do an example

\[
\begin{bmatrix}
e_{1,1} & e_{1,2} & e_{1,3} & 0 \\
e_{2,1} & e_{2,2} & e_{2,3} & 0 \\
e_{3,1} & e_{3,2} & e_{3,3} & 0 \\
e_{4,1} & e_{4,2} & e_{4,3} & 1
\end{bmatrix}
= \begin{bmatrix}
v_{1.x} & v_{2.x} & v_{3.x} & 0 \\
v_{1.y} & v_{2.y} & v_{3.y} & 0 \\
v_{1.z} & v_{2.z} & v_{3.z} & 0 \\
v_{1.t} & v_{2.t} & v_{3.t} & 1
\end{bmatrix}
\]

Where \( t = (0,0,0) \)-\( O \)

\[ = (-2,-2,-2) \]

Camera frame:

- \( V_1 = (0.6, 0, 0.8) \)
- \( V_2 = (0, 1, 0) \)
- \( V_3 = (-0.8, 0, 0.6) \)
- \( O = (2,2,2) \)
Example continued

- Convert Cartesian frame coordinates \((1, 2, 3)\) to Camera frame coordinates

Camera frame:

- \(V_1 = (0.6, 0, 0.8)\)
- \(V_2 = (0, 1, 0)\)
- \(V_3 = (-0.8, 0, 0.6)\)
- \(O = (2, 2, 2)\)

\[
\begin{bmatrix}
0.6 & 0 & -0.8 & 0 \\
0 & 1 & 0 & 0 \\
0.8 & 0 & 0.6 & 0 \\
-2.8 & -2 & 0.4 & 1
\end{bmatrix}
\]
Example continued

- Convert Cartesian frame coordinates \((1, 2, 3, 1)\) to Camera frame coordinates

Camera frame:

\[
\begin{align*}
V_1 &= (0.6, 0, 0.8) \\
V_2 &= (0, 1, 0) \\
V_3 &= (-0.8, 0, 0.6) \\
O &= (2,2,2)
\end{align*}
\]

\[
\begin{bmatrix}
[0.6 & 0 & -0.8 & 0] \\
[0 & 1 & 0 & 0] \\
[0.8 & 0 & 0.6 & 0] \\
[-2.8 & -2 & 0.4 & 1]
\end{bmatrix}
\]

\[
= (0.2, 0, 1.4, 1)
\]

These are the coordinates of \((1,2,3)\) in the Camera Frame.
Example continued

- Let’s convert (0.2, 0, 1.4) back to the Cartesian frame.
- Do we get (1,2,3)?

Camera frame:

V1 = (0.6, 0, 0.8)
V2 = (0, 1, 0)
V3 = (-0.8, 0, 0.6)
O = (2,2,2)
Our goal

World space:
- Triangles in native Cartesian coordinates
- Camera located anywhere

Camera space:
- Camera located at origin, looking down -Z
- Triangle coordinates relative to camera frame

Image space:
- All viewable objects within -1 <= x,y,z <= +1

Screen space:
- All viewable objects within -1 <= x, y <= +1

Device space:
- All viewable objects within 0 <= x <= width, 0 <= y <= height

Discussed this last time, review today

Discuss this today
The viewing transformation is not a combination of simple translations, rotations, scales or shears: it is more complex.
The View Transformation

- Input parameters: \((\alpha, n, f)\)
- Transforms view frustum to image space cube
  - View frustum: bounded by viewing pyramid and planes \(z=-n\) and \(z=-f\)
  - Image space cube: \(-1 \leq u, v, w \leq 1\)

\[
\begin{bmatrix}
\cot(\alpha/2) & 0 & 0 & 0 \\
0 & \cot(\alpha/2) & 0 & 0 \\
0 & 0 & (f+n)/(f-n) & -1 \\
0 & 0 & 2fn/(f-n) & 0
\end{bmatrix}
\]

- Cotangent = \(1/\text{tangent}\)
I personally don’t think it is a good use of class time to derive this matrix.

Well derived at:


But I will do it on Weds if you want to see it.
Let’s do an example

Input parameters: \((\alpha, n, f) = (90, 5, 10)\)

\[
\begin{bmatrix}
\cot(\alpha/2) & 0 & 0 & 0 & 0 \\
0 & \cot(\alpha/2) & 0 & 0 & 0 \\
0 & 0 & (f+n)/(f-n) & -1 & 0 \\
0 & 0 & 2fn/(f-n) & 0 & 0
\end{bmatrix}
\]
Let’s do an example

- Input parameters: \((\alpha, n, f) = (90, 5, 10)\)

\[
\begin{bmatrix}
1 & 0 & 0 & 0 \\
0 & 1 & 0 & 0 \\
0 & 0 & 3 & -1 \\
0 & 0 & 20 & 0
\end{bmatrix}
\]
Let’s do an example

Input parameters: \((\alpha, n, f) = (90, 5, 10)\)

\[
\begin{bmatrix}
1 & 0 & 0 & 0 \\
0 & 1 & 0 & 0 \\
0 & 0 & 3 & -1 \\
0 & 0 & 20 & 0
\end{bmatrix}
\]

Let’s multiply some points:

\((0, 7, -6, 1)\)
\((0, 7, -8, 1)\)
Let’s do an example

**Input parameters:** \(( \alpha, n, f) = (90, 5, 10)\)

\[
\begin{bmatrix}
1 & 0 & 0 & 0 \\
0 & 1 & 0 & 0 \\
0 & 0 & 3 & -1 \\
0 & 0 & 20 & 0 \\
\end{bmatrix}
\]

Let’s multiply some points:

\((0,7,-6,1) = (0,7,-2,6) = (0, 1.16, -0.33)\)

\((0,7,-8,1) = (0,7,4,8) = (0, 0.88, 0.5)\)
Let's do an example

**Input parameters:** \((\alpha, n, f) = (90, 5, 10)\)

\[
\begin{bmatrix}
1 & 0 & 0 & 0 \\
0 & 1 & 0 & 0 \\
0 & 0 & 3 & -1 \\
0 & 0 & 20 & 0
\end{bmatrix}
\]

More points:
- \((0,7,-4,1) = (0,7,-12,4) = (0, 1.75, -3)\)
- \((0,7,-5,1) = (0,7,-15,3) = (0, 2.33, -1)\)
- \((0,7,-6,1) = (0,7,-2,6) = (0, 1.16, -0.33)\)
- \((0,7,-8,1) = (0,7,4,8) = (0, 0.88, 0.5)\)
- \((0,7,-10,1) = (0,7,10,10) = (0, 0.7, 1)\)
- \((0,7,-11,1) = (0,7,13,11) = (0, .63, 1.18)\)
The viewing transformation is not a combination of simple translations, rotations, scales or shears: it is more complex.
More points:

(0, 7, -4, 1) = (0, 7, -12, 4) = (0, 1.75, -3)
(0, 7, -5, 1) = (0, 7, -5, 5) = (0, 1.4, -1)
(0, 7, -6, 1) = (0, 7, -2, 6) = (0, 1.16, -0.33)
(0, 7, -8, 1) = (0, 7, 4, 8) = (0, 0.88, -0.5)
(0, 7, -10, 1) = (0, 7, 10, 10) = (0, 0.7, 1)
(0, 7, -11, 1) = (0, 7, 13, 11) = (0, 0.63, 1.18)

Note there is a non-linear relationship in Z.
Looking forward
... we are almost there

World space:
- Triangles in native Cartesian coordinates
- Camera located anywhere

Camera space:
- Camera located at origin, looking down -Z
- Triangle coordinates relative to camera frame

Image space:
- All viewable objects within -1 <= x,y,z <= +1

Screen space:
- All viewable objects within -1 <= x, y <= +1

Device space:
- All viewable objects within 0 <= x <= width, 0 <= y <= height

Weds:
Discuss final steps and 1F, then on to OpenGL

Discussed this last time, review today

Discussed this today