No Class Tuesday, 1/29

• Will definitely be a YouTube lecture to replace that one.
Office Hours: Weeks 4-10

- Monday: 1-2 (Roscoe)
- Tuesday: 1-2 (Roscoe)
- Wednesday: 1-3 (Roscoe)
- Thursday: 1130-1230 (Hank)
- Friday: 1130-1230 (Hank)
Office Hours: Week 3

- Monday: 415-530 (Hank)
- Tuesday: 1-2, 2-3 (Roscoe)
- Wednesday: 1-3 (Roscoe)
- Thursday: 1130-1230 (Hank)
- Thursday: 1230-230 (Roscoe)
- Friday: 1030-1130 (Hank)
Timeline

- **1C**: due Weds Jan 23rd
- **1D**: assigned today, due Thurs Jan 31st
- **1E**: assigned Thurs Jan 31st, due Weds Feb 6th
  - → will be extra support with this. Tough project.
- **1F**: assigned Feb 7th, due Feb 19th
  - → not as tough as 1E
- **2A**: will be assigned during week of Feb 11th

<table>
<thead>
<tr>
<th>Sun</th>
<th>Mon</th>
<th>Tues</th>
<th>Weds</th>
<th>Thurs</th>
<th>Fri</th>
<th>Sat</th>
</tr>
</thead>
<tbody>
<tr>
<td>Jan 20</td>
<td>Jan 21</td>
<td>Jan 22</td>
<td>Jan 23</td>
<td>Lec 5</td>
<td>Jan 25</td>
<td>Jan 26</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Lec 4</td>
<td>1C due</td>
<td>1D assigned</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Jan 27</td>
<td>Jan 28</td>
<td>Jan 29 (YouTube)</td>
<td>Jan 30</td>
<td>Lec 6</td>
<td>Feb 1</td>
<td>Feb 2</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>1D due</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>1E assigned</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Feb 3</td>
<td>Feb 4</td>
<td>Feb 5 Lec 7</td>
<td>Feb 6</td>
<td>Lec 8</td>
<td>Feb 8</td>
<td>Feb 9</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>1E due</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>1F assigned</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Likely: pre-SuperBowl OH
Great news!!

• No project assignment today...
Project #1D (5%),
Due Thurs Jan 31st

- Goal: interpolation of color and zbuffer
- Extend your project1C code
- File proj1d_geometry.vtk available on web (1.4MB)
- File “reader1d.cxx” has code to read triangles from file.
- No Cmake, project1d.cxx
Color is now floating-point

• We will be interpolating colors, so please use floating point (0 → 1)
• Keep colors in floating point until you assign them to a pixel
• Fractional colors? → use $\text{ceil}(\frac{value}{255})$
  – $\text{ceil}(\frac{value}{255})$
Changes to data structures

class Triangle
{
  public:
    double X[3], Y[3], Z[3];
    double colors[3][3];
};

→ reader1d.cxx will not compile until you make these changes
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
MATH!
Space

- A “space” is a set of points
- Many types of spaces
Here is a space ‘S’:
the points in the blue shape
We can pick an arbitrary point in S and call it our “origin.”
Consider two directions, D1 and D2.
Imagine you live at “O” and you want to get to “X.” Can you do it?

Rules (chess):
- Bishop can only move diagonally
- Rooks can only move in straight lines

Rules (this space):
- You can only move in direction D1 or D2
Imagine you live at “O” and you want to get to “X.” Can you do it?

Rules (chess):
- Bishop can only move diagonally
- Rooks can only move in straight lines

Rules (this space):
- You can only move in direction D1 or D2
Imagine you live at “O” and you want to get to “X.” Can you do it?

Rules (chess):
- Bishop can only move diagonally
- Rooks can only move in straight lines

Rules (this space):
- You can only move in direction D1 or D2
Imagine you live at “O” and you want to get to “X.” Can you do it?

Rules (chess):
- Bishop can only move diagonally
- Rooks can only move in straight lines

Rules (this space):
- You can only move in direction D1 or D2
Imagine you live at “O” and you want to get to “X2.” Can you do it?

Rules (chess):
- Bishop can only move diagonally
- Rooks can only move in straight lines

Rules (this space):
- You can only move in direction D1 or D2
Imagine you live at “O” and you want to get to “X2.” Can you do it?

Rules (chess):
- Bishop can only move diagonally
- Rooks can only move in straight lines

Rules (this space):
- You can only move in direction D1 or D2
Imagine you live at “O” and you want to get to “X3.” Can you do it?

Rules (chess):
- Bishop can only move diagonally
- Rooks can only move in straight lines

Rules (this space):
- You can only move in direction D1 or D2
Imagine you live at “O” and you want to get to “X4.” Can you do it?

Rules (chess):
- Bishop can only move diagonally
- Rooks can only move in straight lines

Rules (this space):
- You can only move in direction D1 or D2
Imagine you live at “O” and you want to get to “X4.” Can you do it?

Rules (chess):
- Bishop can only move diagonally
- Rooks can only move in straight lines

Rules (this space):
- You can only move in direction D1 or D2
Conventions!

- Let \((a, b)\) mean:
  - The number of steps ‘\(a\)’ in direction D1
  - The number of steps ‘\(b\)’ in direction D2
Where is (-3, 2)?

Rules (chess):
- Bishop can only move diagonally
- Rooks can only move in straight lines

Rules (this space):
- You can only move in direction D1 or D2
Paraphrasing Wikipedia:

Let \( B = \{ D_1, D_2 \} \) (a set of two vectors, \( D_1 \) & \( D_2 \))

Let \( S \) be our Shape

\( B \) is a basis for \( S \) if every element of \( S \) can be written as a unique linear combination of elements of \( B \).

The coefficients of this linear combination are referred to as components or coordinates on \( B \) of the vector.

The elements of a basis are called basis vectors.
Why unique?

- Let \((a, b, c)\) mean:
  - The number of steps ‘a’ in direction D1
  - The number of steps ‘b’ in direction D2
  - The number of steps ‘c’ in direction D3

- Then there is more than one way to get to some point \(X\) in \(S\), i.e.,
  - \((a_1, b_1, c_1) = X\) and
  - \((a_2, b_2, c_2) = X\)
What does it mean to form a basis?

- For any vector \( v \), there are unique coordinates \((c_1, \ldots, c_n)\) such that
  \[ v = c_1*v_1 + c_2*v_2 + \ldots + c_n*v_n \]

- Consider some point \( P \).
  - The basis has an origin \( O \)
  - There is a vector \( v \) such that \( O + v = P \)
  - We know we can construct \( v \) using a combination of \( v_i \)’s
  - Therefore we can represent \( P \) in our frame using the coordinates \((c_1, c_2, \ldots, c_n)\)
A basis

- Paraphrasing Wikipedia:
- Let \( B = \{ D_1, D_2 \} \) (a set of two vectors, \( D_1 \) & \( D_2 \))
- Let \( S \) be our Shape
- \( B \) is a **basis** for \( S \) if every element of \( S \) can be written as a **unique** linear combination of elements of \( B \).
- The coefficients of this linear combination are referred to as **components** or **coordinates** on \( B \) of the vector.
- The elements of a basis are called **basis vectors**.
Most common basis

- $D_1 = X$-axis (i.e., $(1,0,0)-(0,0,0)$)
- $D_2 = Y$-axis (i.e., $(0,1,0)-(0,0,0)$)
- $D_3 = Z$-axis (i.e., $(0,0,1)-(0,0,0)$)

Then the coordinate $(2, -3, 5)$ means
- 2 units along $X$-axis
- -3 units along $Y$-axis
- 5 units along $Z$-axis
But we could have other bases

- Instead of “basis 1” (B1)
  - $D_1 = \text{X-axis}$ (i.e., $(1,0,0)$-$(0,0,0)$)
  - $D_2 = \text{Y-axis}$ (i.e., $(0,1,0)$-$(0,0,0)$)
  - $D_3 = \text{Z-axis}$ (i.e., $(0,0,1)$-$(0,0,0)$)

- Use “basis 2” (B2)
  - $D_1 = \text{Y-axis}$ (i.e., $(0,1,0)$-$(0,0,0)$)
  - $D_2 = \text{X-axis}$ (i.e., $(1,0,0)$-$(0,0,0)$)
  - $D_3 = \text{Z-axis}$ (i.e., $(0,0,1)$-$(0,0,0)$)

- Then $(a,b,c)$ in B1 is the same as $(b,a,c)$ in B2
Frame:

- A way to place a coordinate system into a specific location in a space
- Basis + reference coordinate ("the origin")

Cartesian example: (3,4,6)

- It is assumed that we are speaking in reference to the origin location (0,0,0).
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 $(6, 6)$?
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: $(-2, 3)$
Each box is a frame, and each arrow converts to the next frame.

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
Context

- Models stored in “world space” frame
  - Pick an origin, store all points relative to that origin
- We have been rasterizing in “device space” frame
- Our goal: transform from world space to device space
- We will do this using matrix multiplications
  - Multiply point by matrix to convert coordinates from one frame into coordinates in another frame
But wait! There’s more...

- And matrices also useful for more than frame-to-frame conversions.

- So let’s get comfy with matrices.
Matrix

- Defined: a rectangular array of numbers (usually) arranged in rows and columns

- Example
  - 2D matrix
  - “two by three” (two rows, three columns)
    - [3 4 8]
    - [-1 9.2 12]
Matrix: wikipedia picture

A matrix $A$ is an $m$-by-$n$ matrix, where $m$ is the number of rows and $n$ is the number of columns. Each element of the matrix is denoted by $a_{i,j}$, where $i$ is the row index and $j$ is the column index. The matrix can be visualized as:

$$
\begin{bmatrix}
a_{1,1} & a_{1,2} & a_{1,3} & \cdots \\
a_{2,1} & a_{2,2} & a_{2,3} & \cdots \\
a_{3,1} & a_{3,2} & a_{3,3} & \cdots \\
\vdots & \vdots & \vdots & \ddots
\end{bmatrix}
$$
What do you do with matrices?

Lots of things

- Transpose, invert, add, subtract

But most of all: multiply!
Multiplying two 2x2 matrices

\[
\begin{pmatrix}
  a & b \\
  c & d \\
\end{pmatrix}
\times
\begin{pmatrix}
  e & f \\
  g & h \\
\end{pmatrix}
=
\begin{pmatrix}
  (a*e + b*g) & (a*f + b*h) \\
  (c*e + d*g) & (c*f + d*h) \\
\end{pmatrix}
\]
One usage for matrices:
Let \((a, b)\) be the coordinates of a point
Then the 2x2 matrix can transform \((a,b)\) to a new location \(-(a*e+b*g, a*f+b*h)\)
Identity Matrix

\[
\begin{pmatrix}
a & b \\
1 & 0
\end{pmatrix}
\begin{pmatrix}
a \\
b
\end{pmatrix}
= 
\begin{pmatrix}
1 & 0 \\
0 & 1
\end{pmatrix}
\]
\[(a\ b)\quad (2\ 0)\quad (2a\ b)\]
\[X\quad (0\ 1)\quad =\]

Scale in X, not in Y
\[(a \ b) \quad (s \ 0) \quad (sa \ tb)\]

\[X \quad (0 \ t) =\]

Scale in both dimensions
(a, b) \begin{pmatrix} 0 \ -1 \end{pmatrix} \begin{pmatrix} b \ -a \end{pmatrix} = \begin{pmatrix} 1 \ 0 \end{pmatrix} = \begin{pmatrix} (a,b) \ (b,-a) \end{pmatrix}

Rotate 90 degrees counterclockwise
\[
\begin{pmatrix}
a & b \\
-1 & 0
\end{pmatrix}
\begin{pmatrix}
0 & 1 \\
-b & a
\end{pmatrix}
= 
\begin{pmatrix}
-b & a \\
(a, b)
\end{pmatrix}
\]

Rotate 90 degrees counterclockwise
\[
\begin{pmatrix}
a & b \\
\end{pmatrix} \begin{pmatrix}
\cos(\Omega) & -\sin(\Omega) \\
\sin(\Omega) & \cos(\Omega) \\
\end{pmatrix} = \begin{pmatrix}
\cos(\Omega)*a + \sin(\Omega)*b, \\
-sin(\Omega)*a +\cos(\Omega)*b\
\end{pmatrix}
\]

Rotate “\(\Omega\)” degrees counter-clockwise
Combining transformations

How do we rotate by 90 degrees clockwise and then scale X by 2?

Answer: multiply by matrix that multiplies by 90 degrees clockwise, then multiple by matrix that scales X by 2.

But can we do this efficiently?

\[
\begin{pmatrix}
0 & -1 \\
1 & 0
\end{pmatrix}
\begin{pmatrix}
2 & 0 \\
0 & 1
\end{pmatrix}
\begin{pmatrix}
0 & -1 \\
1 & 0
\end{pmatrix}
\begin{pmatrix}
= \\
2 & 0
\end{pmatrix}
\]
How do we scale $X$ by 2 and then rotate by 90 degrees clockwise?

Answer: multiply by matrix that scales $X$ by 2, then multiply by matrix that rotates 90 degrees clockwise.

\[
\begin{pmatrix}
2 & 0 \\
0 & -1
\end{pmatrix}
\begin{pmatrix}
0 & -1 \\
1 & 0
\end{pmatrix}
\begin{pmatrix}
0 & -2 \\
1 & 0
\end{pmatrix}
\]

\[
X =
\begin{pmatrix}
0 & 1 \\
1 & 0
\end{pmatrix}
\begin{pmatrix}
1 & 0 \\
0 & 1
\end{pmatrix}
\begin{pmatrix}
1 & 0 \\
0 & 1
\end{pmatrix}
\]

Rotate then scale
Order matters!!
Translation is harder:

\[
\begin{align*}
(a) & \quad (c) & (a+c) \\
+ & \quad + & = \\
(b) & \quad (d) & (b+d)
\end{align*}
\]

But this doesn’t fit our nice matrix multiply model…

What to do??
Homogeneous Coordinates

\[
\begin{pmatrix}
1 & 0 & 0 \\
0 & 1 & 0 \\
0 & 0 & 1
\end{pmatrix}
\begin{pmatrix}
x \\
y \\
1
\end{pmatrix}
= 
\begin{pmatrix}
x \\
y \\
1
\end{pmatrix}
\]

Add an extra dimension.

A math trick … don’t overthink it.
Homogeneous Coordinates

Translation

We can now fit translation into our matrix multiplication system.
Two really important operations:
- Transform from one frame to another
- Transform geometry (rotate, translate, etc)

Both can be done with matrix operations

In both cases, need homogeneous coordinates

Much of graphics is accomplished via 4x4 matrices
- And: you can compose the matrices and do bunches of things at once (EFFICIENCY)
<table>
<thead>
<tr>
<th><strong>Former type</strong></th>
<th>Public</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Traded as</strong></td>
<td>NYSE: SGI</td>
</tr>
<tr>
<td></td>
<td>OTC Pink: SGID.pk</td>
</tr>
<tr>
<td></td>
<td>NASDAQ: SGIC</td>
</tr>
<tr>
<td><strong>Industry</strong></td>
<td>Computer hardware and software</td>
</tr>
<tr>
<td><strong>Fate</strong></td>
<td>Chapter 11 bankruptcy; assets acquired by Rackable Systems, which renamed itself Silicon Graphics International Corp.</td>
</tr>
<tr>
<td><strong>Founded</strong></td>
<td>November 9, 1981; 37 years ago Mountain View, California, U.S.¹</td>
</tr>
<tr>
<td><strong>Defunct</strong></td>
<td>May 11, 2009</td>
</tr>
<tr>
<td><strong>Headquarters</strong></td>
<td>Sunnyvale, California, U.S.</td>
</tr>
<tr>
<td><strong>Key people</strong></td>
<td>Jim Clark, Kurt Akeley, Ed McCracken, Thomas Jermoluk</td>
</tr>
<tr>
<td><strong>Products</strong></td>
<td>High-performance computing, visualization and storage</td>
</tr>
<tr>
<td><strong>Website</strong></td>
<td><a href="http://www.sgi.com/">www.sgi.com/</a></td>
</tr>
</tbody>
</table>
3dfx Voodoo
(source: wikipedia)
Early GPUs

- Special hardware to do 4x4 matrix operations
- A lot of them (in parallel)
GPUs now

- Many, many, many cores
- Each code less powerful than typical CPU core
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 $-1 \leq x, y, z \leq 1$

Screen space:
- All viewable objects within $-1 \leq x, y \leq 1$

Device space:
- All viewable objects within $0 \leq x \leq \text{width}$,
  $0 \leq y \leq \text{height}$
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
World Space

- World Space is the space defined by the user’s coordinate system.
- This space contains the portion of the scene that is transformed into image space by the camera transform.
- Many of the spaces have “bounds”, meaning limits on where the space is valid.
- With world space 2 options:
  - No bounds
  - User specifies the bound
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

Camera Transform
World Space

- World Space is the space defined by the user’s coordinate system.
- This space contains the portion of the scene that is transformed into image space by the camera transform.
- Many of the spaces have “bounds”, meaning limits on where the space is valid.
- With world space 2 options:
  - No bounds
  - User specifies the bound
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
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];
};
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 
  \(-1 \leq x, y, z \leq +1\)

Screen space:
- All viewable objects within 
  \(-1 \leq x, y \leq +1\)

Device space:
- All viewable objects within 
  \(0 \leq x \leq \text{width}, 0 \leq y \leq \text{height}\)
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
Image Space

- Image Space is the three-dimensional coordinate system that contains screen space.
- It is the space where the camera transformation directs its output.
- The bounds of Image Space are 3-dimensional cube.

\[ \{(x, y, z) : -1 \leq x \leq 1, -1 \leq y \leq 1, -1 \leq z \leq 1\} \]

(or \(-1 \leq z \leq 0\))
Image Space Diagram

X = -1
Y = -1
Z = 1
X = 1
Y = 1
Z = -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

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
Screen Space

- Screen Space is the intersection of the xy-plane with Image Space.
- Points in image space are mapped into screen space by projecting via a parallel projection, onto the plane $z = 0$.
- Example:
  - A point $(0, 0, z)$ in image space will project to the center of the display screen.
Screen Space Diagram

+1

Y

-1

-1 X +1

[Diagram showing a color-coded map with X and Y axes and labeled with +1, -1, and 0 points.]
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
Device Space

- Device Space is the lowest level coordinate system and is the closest to the hardware coordinate systems of the device itself.

- Device space is usually defined to be the $n \times m$ array of pixels that represent the area of the screen.

- A coordinate system is imposed on this space by labeling the lower-left-hand corner of the array as $(0,0)$, with each pixel having unit length and width.
Device Space Example

- **Pixel (0, 0)**
- **Pixel (3, 7)**
- **Pixel (15, 15)**
Device Space With Depth Information

- Extends Device Space to three dimensions by adding z-coordinate of image space.
- Coordinates are \((x, y, z)\) with
  
  \[
  0 \leq x \leq n \\
  0 \leq y \leq m \\
  z \text{ arbitrary (but typically } -1 \leq z \leq +1 \text{ or } -1 \leq z \leq 0 \text{)}
  \]
Easiest Transform

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
(x, y, z) \rightarrow (x', y', z'), \text{ where}

- x' = n*(x+1)/2
- y' = m*(y+1)/2
- z' = z

(for an n \times m image)

Matrix:

\[
\begin{pmatrix}
  x' & 0 & 0 & 0 \\
  0 & y' & 0 & 0 \\
  0 & 0 & z' & 0 \\
  0 & 0 & 0 & 1
\end{pmatrix}
\]
Coming Up…

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