Misc.

- How to deal with pixels outside the screen
- How to get geometry in the first place?!?
I will grade all 1B’s very soon

I do not plan to grade again until 1F

I will grade anyone’s project if there is a good reason

Rubrics:
- I really intend for you to get “0 pixels difference”
- Anything less than that can be very problematic to grade.
Everyone understand what to do?

OH today 3-4

Keep in mind:

```c
for (int j = leftIdx; j <= rightIdx; j++)
{
    if (j < 0 || j >= width)
        continue;
    double proportion;
    if (rightPos != leftPos)
        proportion = (((double))leftPos) / (rightPos - leftPos);
    else
        proportion = 1.0;
    unsigned char rgb[3];
    rgb[0] = (unsigned char) ceil441(255.0*(leftRGB[0] + proportion*rightRGB[0]));
    rgb[1] = (unsigned char) ceil441(255.0*(leftRGB[1] + proportion*rightRGB[1]));
    rgb[2] = (unsigned char) ceil441(255.0*(leftRGB[2] + proportion*rightRGB[2]));
    double z = leftZ + proportion*(rightZ-leftZ);
    Assign(j, i, rgb, z);
}
Outline

- Lighting Review
- Basic Transformations
- Arbitrary Camera Positions
Outline

- Lighting Review
- Basic Transformations
- Arbitrary Camera Positions
What is a dot product?

- \( \mathbf{A} \cdot \mathbf{B} = \mathbf{A}.x \mathbf{B}.x + \mathbf{A}.y \mathbf{B}.y \)
  - (or \( \mathbf{A}.x \mathbf{B}.x + \mathbf{A}.y \mathbf{B}.y + \mathbf{A}.z \mathbf{B}.z \))

- Physical interpretation:
  - \( \mathbf{A} \cdot \mathbf{B} = \cos(\alpha) / (||\mathbf{A}|| \cdot ||\mathbf{B}||) \)

\( (\mathbf{B}.x, \mathbf{B}.y) \)  \( (\mathbf{A}.x, \mathbf{B}.y) \)  \( \alpha \)
What is the cross product?

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

- What is the physical interpretation of a cross product?
  - Finds a vector perpendicular to both $A$ and $B$. 

*[Image of a football field]*
Norm, Normal, Normalize, Oh My!

- **Norm**: the length of a vector ($|\|A\||$)

- **Normal**: a perpendicular vector to a plane coincident with geometry

- **Normalize**: the operation to create a vector with length 1 ($A/\|\|A\||$)

- All 3 are important for this class
Two ways to treat normals:

- Constant over a triangle
- Varying over a triangle

- Constant over a triangle $\leftrightarrow$ flat shading
- Varying over a triangle $\leftrightarrow$ smooth shading
Flat vs Smooth Shading
Lighting and Normals

- Two ways to treat normals:
  - Constant over a triangle
  - Varying over a triangle

- Constant over a triangle $\longleftrightarrow$ flat shading
  - Take $(C-A) \times (B-A)$ as normal over whole triangle

- Varying over a triangle $\longleftrightarrow$ smooth shading
  - Calculate normal at vertex, then use linear interpolation
    - How do you calculate normal at a vertex?
    - How do you linearly interpolate normals?
LERPingo vectors

- LERP = Linear Interpolate
- Goal: interpolate vector between A and B.
- Consider vector X, where \( X = B - A \)
- Back to normal LERP:
  - \( A + t*(B-A) = A + t*X \)
- You will need to LERP vectors for 1E
Our goal:
- For each pixel, calculate a shading factor
- Shading factor typically between 0 and 1, but sometimes >1
  - Shading >1 makes a surface more white

3 types of lighting to consider:
- Ambient
  - Light everywhere
- Diffuse
  - Rough surface
- Specular
  - Smooth surface

Our game plan: Calculate all 3 and combine them.
How to handle shading values greater than 1?

- Color at pixel = (1.0, 0.4, 0.8)
- Shading value = 0.5
  - Easy!
  - Color = (0.5, 0.2, 0.4) → (128, 52, 103)
- Shading value = 2.0
  - Color = (1.0, 0.8, 1.0) → (255, 204, 255)
- Color_R = 255*min(1, R*shading_value)
- This is how specular makes things whiter and whiter.
  - But it won’t put in colors that aren’t there.
Ambient Lighting

- Ambient light
  - Same amount of light everywhere in scene
  - Can model contribution of many sources and reflecting surfaces

Surface lit with ambient lighting only
Lambertian Surface

- Perfectly diffuse reflector
- Light scattered equally in all directions

Extreme zoom-in of part of a diffuse surface … light is scattered in all directions

(this image shows 5 of the directions)

Slide inspired by Ed Angel Computer Graphics Book
How much light should be reflected in this case?

A: $\cos(\alpha)$
And note that:
$\cos(0) = 1$
$\cos(90) = 0$
Diffuse Lighting

Lambertian surfaces reflect light equally in all directions.

How much light makes it to viewer V1? Viewer V2?

A: \( \cos(\alpha) \) for both ... = \( L \cdot N \)

Lambertian surfaces reflect light equally in all directions.
Diffuse Lighting

- Diffuse light
  - Light distributed evenly in all directions, but amount of light depends on orientation of triangles with respect to light source.
  - Different for each triangle

Surface lit with diffuse lighting only
What about cases where $\mathbf{L} \cdot \mathbf{N} < 0$?
What about cases where \( L \cdot N < 0 \)?

\[ L \cdot N = -1 \]

Non-sensical ... takes away light?

Common solution:

\[ \text{Diffuse light} = \max(0, L \cdot N) \]
But wait...

If you have an open surface, then there is a “back face”. The back face has the opposite normal.

How can we deal with this case?

Idea #1: encode all triangles twice, with different normals
Idea #2: modify diffuse lighting model

Diffuse light = abs(L⋅N)

This is called two-sided lighting
Specular Lighting

Light reflects in all directions. But the surface is smooth, not Lambertian, so amount of reflected light varies. So how much light??
How much light reflects with specular lighting?

Shading = strength * (V·R)^shiny factor.

But what is R?

It is a formula: \( R = 2(L·N)N - L \)
Two-sided lighting

- For specular lighting, we will use one-sided lighting for project 1E
  - It just looks better

- Diffuse: $\text{abs}(L \cdot N)$
- Specular: $\max(0, S^*(R \cdot V)^r)$
Outline

- Math Basics
- Lighting Basics
- The Phong Model
Phong Model

- Combine three lighting effects: ambient, diffuse, specular
Phong Model

- Simple version: 1 light, with “full intensity” (i.e., don’t add an intensity term)

- Phong model
  - $\text{Shading\_Amount} = K_a + K_d \times \text{Diffuse} + K_s \times \text{Specular}$

- Signature:
  - double CalculatePhongShading(LightingParameters &, double *viewDirection, double *normal)
  - For us, viewDirection = (0, 0, +1)
struct LightingParameters
{
    LightingParameters(void)
    {
        lightDir[0] = -0.6;
        lightDir[1] = 0;
        lightDir[2] = -0.8;
        Ka = 0.3;
        Kd = 0.7;
        Ks = 5.3;
        alpha = 7.5
    }
};

double lightDir[3]; // The direction of the light source
double Ka;          // The coefficient for ambient lighting.
double Kd;          // The coefficient for diffuse lighting.
double Ks;          // The coefficient for specular lighting.
double Ks;          // The exponent term for specular lighting.
};
Let’s do an example

- **Diffuse**: \( \text{abs}(L \cdot N) \)
- **Specular**: \( \max(0, S \cdot (R \cdot V)^\gamma) \)
- \( R = 2 \cdot (L \cdot N) \cdot N - L \)

- **Shading_Amount** = 
  \[ K_a + K_d \cdot \text{Diffuse} + K_s \cdot \text{Specular} \]

```c
struct LightingParameters
{
  LightingParameters(void)
  {
    lightDir[0] = -0.6;
    lightDir[1] = 0;
    lightDir[2] = -0.8;
    Ka = 0.3;
    Kd = 0.7;
    Ks = 5.3;
    alpha = 7.5
  }
};
```

Normal = (0, 0, 1)
Color = (255, 0, 128)
View = (0, 0, +1)
- Goal: add Phong shading
- Extend your project1D code
- File proj1egeometry.vtk available on web (9MB)
- File “reader1e.cxx” has code to read triangles from file.
- No Cmake, project1e.cxx
Changes to data structures

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

➔ reader1e.cxx will not compile until you make these changes
➔ reader1e.cxx will initialize normals at each vertex
More comments

- New: more data to help debug
  - I will make the shading value for each pixel available.
  - I will also make it available for ambient, diffuse, specular.
- Don’t forget to do two-sided lighting
- This project in a nutshell:
  - LERP normal to a pixel
    - You all are great at this now!!
  - Add method called “CalculateShading”.
    - My version of CalculateShading is about ten lines of code.
  - Modify RGB calculation to use shading.
Outline

- Lighting Review
- Basic Transformations
- Arbitrary Camera Positions
MP = P’

Matrix M transforms point P to make new point P’.

\[
\begin{pmatrix}
  a & b \\
  c & d \\
\end{pmatrix}
\begin{pmatrix}
  x \\
  y \\
\end{pmatrix}
=
\begin{pmatrix}
  a\times x + b\times y \\
  c\times x + d\times y \\
\end{pmatrix}
\]

\(M\) takes point \((x,y)\) to point \((a\times x+b\times y, c\times x+d\times y)\)
MP = P'
Matrix M transforms point P to make new point P.

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

Identity Matrix
MP = P'
Matrix M transforms point P to make new point P.

\[
\begin{pmatrix}
2 & 0 \\
0 & 1
\end{pmatrix}
\begin{pmatrix}
x \\
y
\end{pmatrix}
= 
\begin{pmatrix}
2x \\
y
\end{pmatrix}
\]

Scale in X, not in Y
MP = P’
Matrix M transforms point P to make new point P.

\[
\begin{pmatrix}
  s & 0 \\
  0 & t
\end{pmatrix}
\begin{pmatrix}
  x \\
  y
\end{pmatrix}
= 
\begin{pmatrix}
  sx \\
  ty
\end{pmatrix}
\]

Scale in both dimensions
MP = P’
Matrix M transforms point P to make new point P.

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

Switch X and Y
MP = P’
Matrix M transforms point P to make new point P.

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

Rotate 90 degrees clockwise
MP = P’
Matrix M transforms point P to make new point P.

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

Rotate 90 degrees counterclockwise
MP = P’
Matrix M transforms point P to make new point P.

\[
\begin{pmatrix}
\cos(a) & \sin(a) \\
-sin(a) & \cos(a)
\end{pmatrix}
\begin{pmatrix}
x \\
y
\end{pmatrix}
=
\begin{pmatrix}
\cos(a)x + \sin(a)y \\
-\sin(a)x + \cos(a)y
\end{pmatrix}
\]

Rotate “a” 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?**

\[
\begin{pmatrix}
0 & 1 \\
-1 & 0 \\
\end{pmatrix}
\begin{pmatrix}
2 & 0 \\
0 & 1 \\
\end{pmatrix}
= 
\begin{pmatrix}
0 & 1 \\
-2 & 0 \\
\end{pmatrix}
\]
Reminder: matrix multiplication

\[
\begin{pmatrix}
    a & b \\
    c & d
\end{pmatrix}
\begin{pmatrix}
    e & f \\
    g & h
\end{pmatrix}
= 
\begin{pmatrix}
    (a*e+b*g) & (a*f+b*h) \\
    (c*e+d*g) & (c*f+d*h)
\end{pmatrix}
\]
Combining transformations

- How do we rotate by 90 degrees clockwise and then scale X by 2?
  - Answer: multiply by matrix that rotates by 90 degrees clockwise, then multiply by matrix that scales X by 2.
  - But can we do this efficiently?

\[
\begin{pmatrix} 0 & 1 \\ -1 & 0 \end{pmatrix} \begin{pmatrix} 2 & 0 \\ 0 & 1 \end{pmatrix} \begin{pmatrix} 0 & 1 \\ -2 & 0 \end{pmatrix} = X
\]
Combining transformations

- 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.

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

Multiply then scale
Order matters!!
Translation is harder:

\[(a) + (c) = (a+c)\]
\[(b) + (d) = (b+d)\]

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

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

Translation

We can now fit translation into our matrix multiplication system.
Outline

- Lighting Review
- Basic Transformations
- Arbitrary Camera Positions
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...
  - \(3x(1,0,0)\)
  - \(4x(0,1,0)\)
  - \(6x(0,0,1)\)

- 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.

- What is a basis?? ← linear algebra term
What does it mean to form a basis?

- For any vector \( \mathbf{v} \), there are unique coordinates \((c_1, ..., c_n)\) such that

\[
\mathbf{v} = c_1*\mathbf{v}_1 + c_2*\mathbf{v}_2 + ... + c_n*\mathbf{v}_n
\]
What does it mean to form a basis?

- For any vector $v$, there are unique coordinates $(c_1, \ldots, c_n)$ such that
  \[ v = c_1v_1 + c_2v_2 + \ldots + c_nv_n \]

- 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 $(c_1, c_2, \ldots, c_n)$
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: $( -2, 3)$
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