Computer Graphics
CMU 15-462/662

Lecture 1:
Course Intro:
Overview
TODAY: Overview Computer Graphics

- Two main objectives:
  - Try to understand broadly what computer graphics is about
  - “Implement” our 1st algorithm for making images of 3D shapes

- Note: all logistics on course webpage
Q: What is computer graphics?
Probably an image like this comes to mind:
Q: ...ok, but more fundamentally: What is computer graphics—and why do we need it?
Early computer (ENIAC), 1945

punch card (~120 bytes)
There must be a better way!
Credit: PC World, "A Brief History of Computer Displays"
Sketchpad (Ivan Sutherland, 1963)
Coming down the pipe...

2020 virtual reality headset: 2x 2160x2160 @ 90Hz => 2.3GB/s
Why *visual* information?

About 30% of brain dedicated to visual processing...

...eyes are highest-bandwidth port into the head!
What is computer graphics?

**com•put•er** **graph•ics** /kəmˈpyōdər ˈɡrafiks/ *n.*
The use of computers to synthesize visual information.
What is computer graphics?

The use of computers to synthesize visual information.

Why only visual?
Graphics has evolved a lot since its early days... no longer just about turning on pixels!
Turning digital information into sensory stimuli

computer graphics /kəmˈpyʊdər ˈɡræfɪks/ n.
The use of computers to synthesize and manipulate sensory information.

(sound)

(touch)

(...What about taste? Smell?!)
Turning digital information into physical matter
Definition of Graphics, Revisited

**computer graphics** /kəmˈpyʊdər ˈɡrafɪks/ *n.*

The use of computation to turn digital information into sensory stimuli.
Even this definition is too narrow...
SIGGRAPH 2020 Technical Papers Trailer
Computer graphics is *everywhere!*
Entertainment (movies, games)
Entertainment

- Not just cartoons!
Art and design
Industrial design
Computer aided engineering (CAE)
Architecture
Scientific/mathematical visualization
Medical/anatomical visualization
Navigation
Communication
Foundations of computer graphics

- All these applications demand **sophisticated** theory & systems
- **Theory**
  - basic representations *(how do you digitally encode shape, motion?)*
  - sampling & aliasing *(how do you acquire & reproduce a signal?)*
  - numerical methods *(how do you manipulate signals numerically?)*
  - radiometry & light transport *(how does light behave?)*
  - perception *(how does this all relate to humans?)*
  - …
- **Systems**
  - parallel, heterogeneous processing
  - graphics-specific programming languages
  - …
ACTIVITY: modeling and drawing a cube

- Goal: generate a realistic drawing of a cube
- Key questions:
  - *Modeling:* how do we describe the cube?
  - *Rendering:* how do we then visualize this model?
ACTIVITY: modeling the cube

- Suppose our cube is...
  - centered at the origin (0,0,0)
  - has dimensions 2x2x2
  - edges are aligned with x/y/z axes

QUESTION: What are the coordinates of the cube vertices?

A: (1, 1, 1)   E: (1, 1,-1)
B: (-1, 1, 1)  F: (-1, 1,-1)
C: (1,-1, 1)   G: (1,-1,-1)
D: (-1,-1, 1)  H: (-1,-1,-1)

QUESTION: What about the edges?

AB, CD, EF, GH, AC, BD, EG, FH, AE, CG, BF, DH
ACTIVITY: drawing the cube

- Now have a digital description of the cube:

  **VERTICES**
  
  A: (1, 1, 1)  E: (1, 1, -1)
  B: (-1, 1, 1)  F: (-1, 1, -1)
  C: (1, -1, 1)  G: (1, -1, -1)
  D: (-1, -1, 1)  H: (-1, -1, -1)

  **EDGES**
  
  AB, CD, EF, GH,
  AC, BD, EG, FH,
  AE, CG, BF, DH

- How do we draw this 3D cube as a 2D (flat) image?

- Basic strategy:
  1. map 3D vertices to 2D points in the image
  2. connect 2D points with straight lines

- ...Ok, but how?
Perspective projection

- Objects look smaller as they get further away ("perspective")
- Why does this happen?
- Consider simple ("pinhole") model of a camera:
**Perspective projection: side view**

- Where exactly does a point \( p = (x, y, z) \) end up on the image?
- Let’s call the image point \( q = (u, v) \)
Perspective projection: side view

- Where exactly does a point $p = (x, y, z)$ end up on the image?
- Let’s call the image point $q = (u, v)$
- Notice two similar triangles:

- Assume camera has unit size, origin is at pinhole $c$
- Then $v/1 = y/z$, i.e., vertical coordinate is just the slope $y/z$
- Likewise, horizontal coordinate is $u = x/z$
ACTIVITY: now draw it!

- Repeat the same simple algorithm 12 times
  - Once for each edge
  - Assume camera is at $c=(2,3,5)$
  - Convert $(X,Y,Z)$ of both endpoints to $(u,v)$:
    1. subtract camera $c$ from vertex $(X,Y,Z)$ to get $(x,y,z)$
    2. divide $(x,y)$ by $z$ to get $(u,v)$—*write as a fraction*
  - Draw line between $(u1,v1)$ and $(u2,v2)$

**VERTICES**

<p>| | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>( 1, 1, 1 )</td>
<td>E</td>
</tr>
<tr>
<td>B</td>
<td>(-1, 1, 1 )</td>
<td>F</td>
</tr>
<tr>
<td>C</td>
<td>( 1,-1, 1 )</td>
<td>G</td>
</tr>
<tr>
<td>D</td>
<td>(-1,-1, 1 )</td>
<td>H</td>
</tr>
</tbody>
</table>

**EDGES**

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>AB</td>
<td>CD</td>
</tr>
<tr>
<td>CD</td>
<td>EF</td>
</tr>
<tr>
<td>EDGES</td>
<td>GH</td>
</tr>
<tr>
<td>FH</td>
<td>AC</td>
</tr>
<tr>
<td>BD</td>
<td>EG</td>
</tr>
<tr>
<td>FH</td>
<td>AE</td>
</tr>
<tr>
<td>CG</td>
<td>BF</td>
</tr>
<tr>
<td>DH</td>
<td></td>
</tr>
</tbody>
</table>
ACTIVITY: output on graph paper
ACTIVITY: How did you do?

2D coordinates:
A: 1/4, 1/2
B: 3/4, 1/2
C: 1/4, 1
D: 3/4, 1
E: 1/6, 1/3
F: 1/2, 1/3
G: 1/6, 2/3
H: 1/2, 2/3
ACTIVITY: Previous year’s result
Success! We turned purely digital information into purely visual information, using a completely algorithmic procedure.
But wait...

How do we draw lines on a computer?
Close up photo of pixels on a modern display

Lorem
Output for a raster display

- Common abstraction of a raster display:
  - Image represented as a 2D grid of “pixels” (picture elements) **
  - Each pixel can take on a unique color value

** We will strongly challenge this notion of a pixel “as a little square” soon enough. But let’s go with it for now. ;-)
What pixels should we color in to depict a line?

“Rasterization”: process of converting a continuous object to a discrete representation on a raster grid (pixel grid)
What pixels should we color in to depict a line?

Light up all pixels intersected by the line?
What pixels should we color in to depict a line?

Diamond rule (used by modern GPUs):
light up pixel if line passes through associated diamond
What pixels should we color in to depict a line?

Is there a right answer?
(consider a drawing a “line” with thickness)
How do we find the pixels satisfying a chosen rasterization rule?

- Could check every single pixel in the image to see if it meets the condition...
  - $O(n^2)$ pixels in image vs. at most $O(n)$ “lit up” pixels
  - *must* be able to do better! (e.g., work proportional to number of pixels in the drawing of the line)
Incremental line rasterization

- Let’s say a line is represented with integer endpoints: \((u_1,v_1), (u_2,v_2)\)
- Slope of line: \(s = (v_2-v_1) / (u_2-u_1)\)
- Consider an easy special case:
  - \(u_1 < u_2, v_1 < v_2\) (line points toward upper-right)
  - \(0 < s < 1\) (more change in x than y)

\[
v = v_1;
\text{for}(u=u_1; \ u<=u_2; \ u++)
\{
    v += s;
    \text{draw}(u, \ \text{round}(v))
\}
\]

Easy to implement... not how lines are drawn in modern software/hardware!
We now have our first complete graphics algorithm!

Digital information

<table>
<thead>
<tr>
<th>VERTICES</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>A: (1, 1, 1)</td>
<td></td>
</tr>
<tr>
<td>B: (-1, 1, 1)</td>
<td></td>
</tr>
<tr>
<td>C: (1, -1, 1)</td>
<td></td>
</tr>
<tr>
<td>D: (-1, -1, 1)</td>
<td></td>
</tr>
<tr>
<td>E: (1, 1, -1)</td>
<td></td>
</tr>
<tr>
<td>F: (-1, 1, -1)</td>
<td></td>
</tr>
<tr>
<td>G: (1, -1, -1)</td>
<td></td>
</tr>
<tr>
<td>H: (-1, -1, -1)</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>EDGES</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>AB, CD, EF, GH,</td>
<td></td>
</tr>
<tr>
<td>AC, BD, EG, FH,</td>
<td></td>
</tr>
<tr>
<td>AE, CG, BF, DH</td>
<td></td>
</tr>
</tbody>
</table>

Visual information

This is fundamentally what computer graphics is all about…
So far, just made a simple line drawing of a cube.

For more realistic pictures, will need a much richer model of the world:

GEOMETRY
MATERIALS
LIGHTS
CAMERAS
MOTION

... 

Will see all of this (and more!) as our course progresses.
Learn by making/doing!

- Build up “Scotty3D” package for modeling/rendering/animation

Broken up into four major assignments...
Assignment 1: Rasterization
Motivation: display images like these!

[sources: Charles Williams, @xlavoc on codepen.io]
Assignment 2: Geometric Modeling
Motivation: create models like these!

[sources: Richard Yot, 3D-Ace, contrafibbularities, 3ddd.ru]
Assignment 3: Photorealistic Rendering
Motivation: render images like these!

WALL-E (Pixar 2009)

Moana (Disney 2016)

Lucas Lira (2020)
Assignment 4: Animation

(cribbed from Alec Jacobson)
Motivation: make animations like these!

Stephen Candell / Sony Pictures Imageworks (2017)

Yans Media (2015)

Pixar (2016)

Autonomous Systems Lab (2016)
See you next time!

- Before diving in, we’ll do a math review & preview
  - Linear algebra, vector calculus
  - Help make the rest of the course easier!