Computer Graphics Lecture 02: Introduction to 2D and 3D Graphics

DR. ELVIS S. LIU
SOUTHERN UNIVERSITY OF SCIENCE AND TECHNOLOGY
SPRING 2018

OpenGL

LECTURE 02: INTRODUCTION TO 2D AND 3D GRAPHICS

OpenGL (in the past)

- Created by Silicon Graphics Inc. (SGI, http://sgi.com) in 1992, now managed by the non-profit Khronos Group (http://khronos.org)
- Originally aimed to allow any OpenGL program to run on a variety of graphics hardware devices
- Invented when “fixed-function” hardware was the norm
  - Techniques were implemented in the hardware; OpenGL calls sent commands to the hardware to activate / configure different features

OpenGL (today)

- Now supports programmable hardware – the common industry practice today
  - Modern graphics cards are miniature, highly parallel computers themselves, with many-core GPUs, on-board RAM, etc.
  - GPUs are a large collection of highly parallel high speed arithmetic units; several thousand cores
  - GPUs run simple programs (called “shaders”): take in vertices and other data and output a color value for an individual pixel.
    - GLSL, (O)GL Shader Language, is C-like language, controls arithmetic pipelines
    - Other shader languages: (DirectX) High-Level Shader Language, RenderMan Shading Language for offline rendering
  - Implement new features in shaders instead of waiting for hardware vendors to support them in h/w

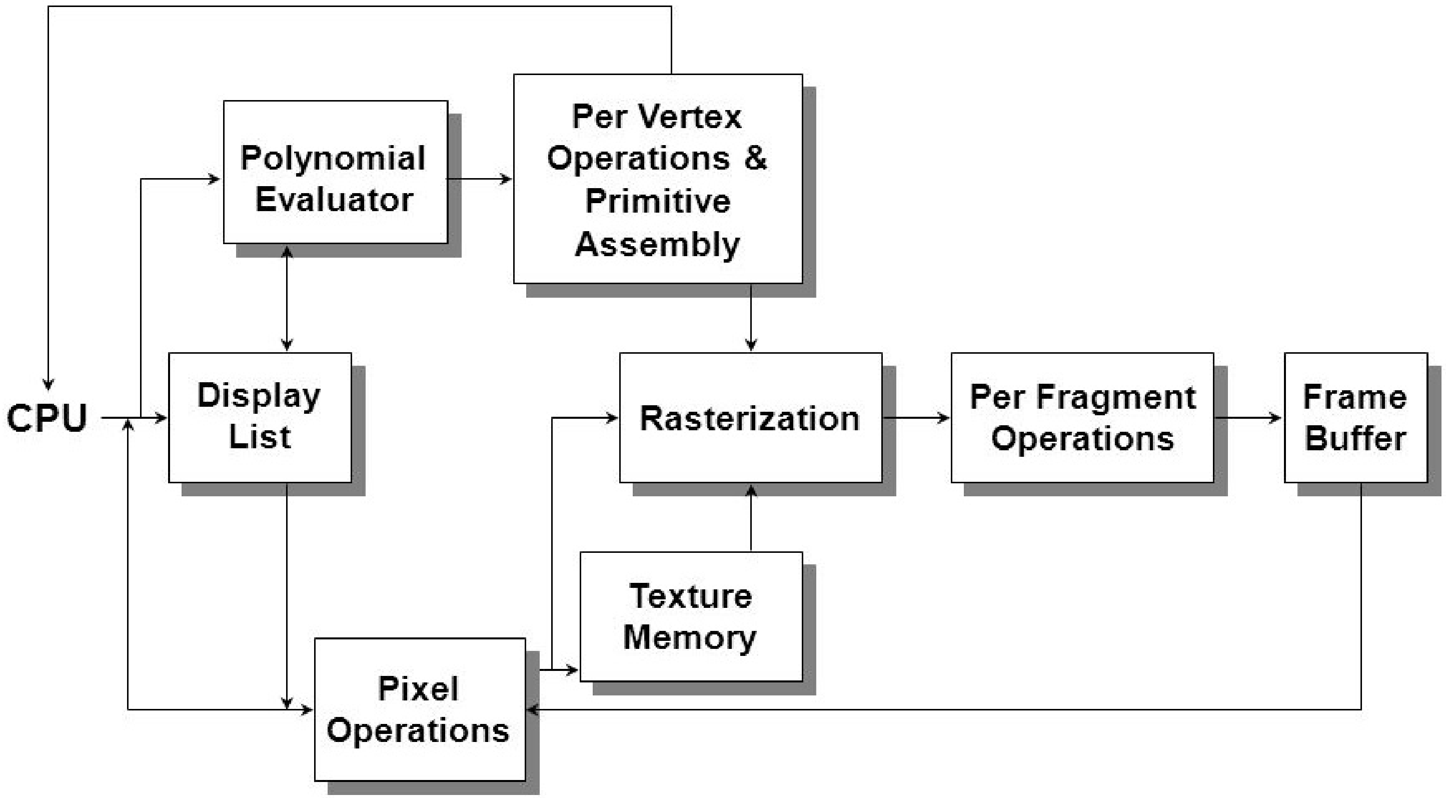
OpenGL

- Immediate-mode graphics API
  - No display model, application must direct OpenGL to draw primitives
- Implemented in C, also works in C++
  - Bindings available for many other programming languages
- Cross-platform
  - Also available on mobile (OpenGL EL) and in the browser (WebGL)
  - Different platforms provide ‘glue’ code for initializing OpenGL within the desktop manager (e.g. GLX, WGL)

Why OpenGL for 3D?

- Widely used in industry and academia for interactive or real-time 3D graphics
- Old fixed-function API (OpenGL 1.x) assisted rapid prototyping of simple 3D scenes with “classical” lighting effects
  - Experiment with simple ideas quickly
- Modern programmable API allows for more flexibility and control

OpenGL Architecture

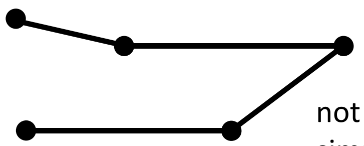


Representation of Shapes

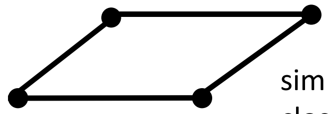
LECTURE 02: INTRODUCTION TO 2D AND 3D GRAPHICS

2D Shapes

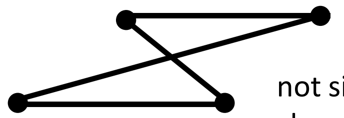
- Lines and polylines:
  - Polylines: lines drawn between ordered points
  - A closed polyline is a polygon, a simple polygon has no self-interactions



not closed,
simple polyline

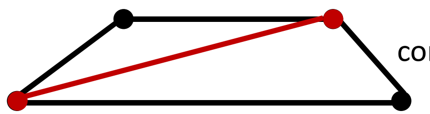


simple polygon and
closed polyline

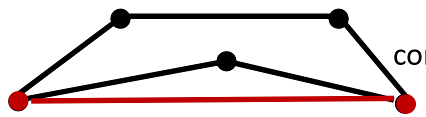


not simple polygon,
closed polyline

- Convex and concave polygons
  - Convex: Line between any two points is inside polygon
  - Concave: At least one line between two points crosses outside polygon



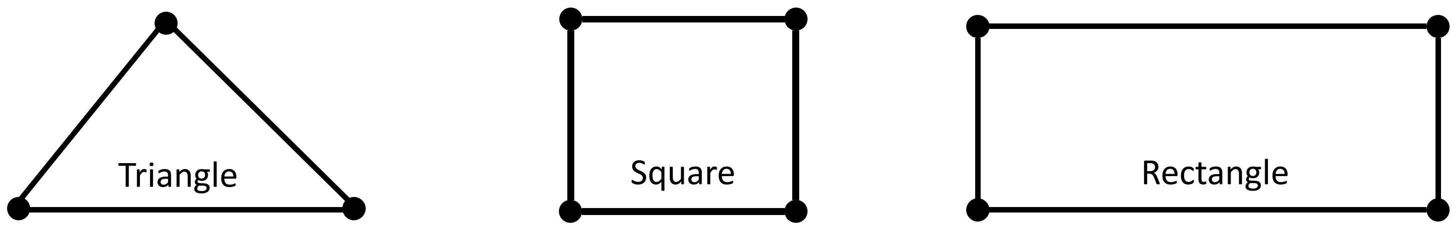
convex



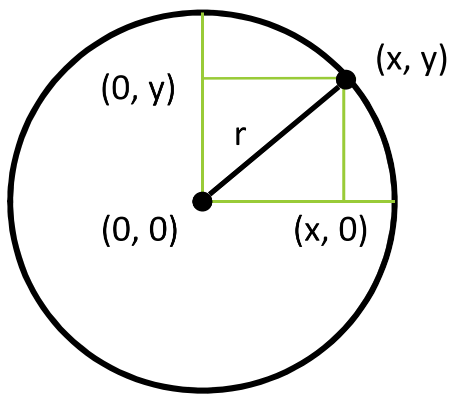
concave

2D Shapes (cont.)

- Special Polygons



- Circles:
  - Set of all points equidistant from one point called centre
  - The distance from the centre is the radius r
  - The equation from a circle centred at (0,0) is
    - $r^2 = x^2 + y^2$

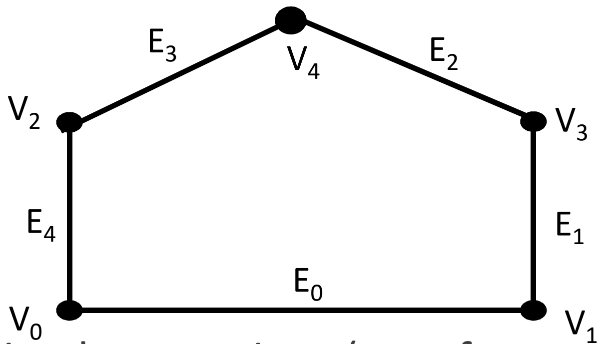


Representing Shapes

- Vertex and edge tables:
  - General purpose, minimal overhead, reasonably efficient
  - Each vertex listed once
  - Each edge is an ordered pair of indices to the vertex list

Vertices	
0	(0, 0)
1	(2, 0)
2	(0, 1)
3	(2, 1)
4	(1, 1.5)

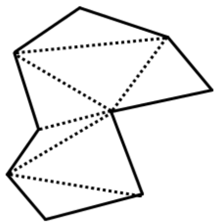
Edges	
0	(0, 1)
1	(1, 3)
2	(3, 4)
3	(4, 2)
4	(2, 0)



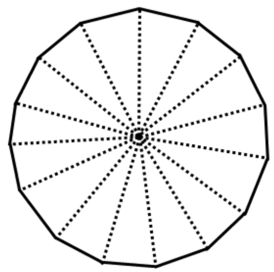
  - Sufficient to draw shape and perform simple operations (transforms, point inside/outside)
  - Edges listed in counterclockwise winding order for consistency with 3D where we need to compute outward-facing normals

3D Shapes

- 3D shapes are usually represented as a collection of vertices that make up triangles or quads
  - OpenGL uses triangles
  - Other methods include 3D voxels, polynomial splines, etc.
- A polygon is a plane figure that is bounded by a finite chain of straight line segments closing in a loop to form a closed polygonal chain or circuit.
- We can use triangles to build arbitrary polygons, and approximate smooth shapes.

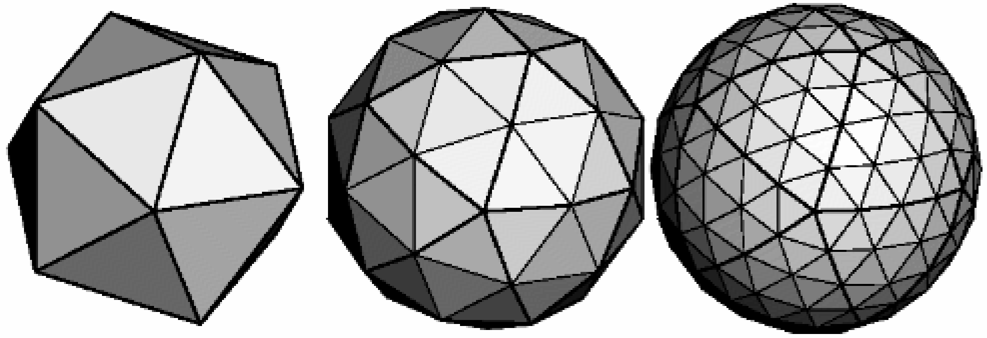


A complex polygon made of triangle primitives

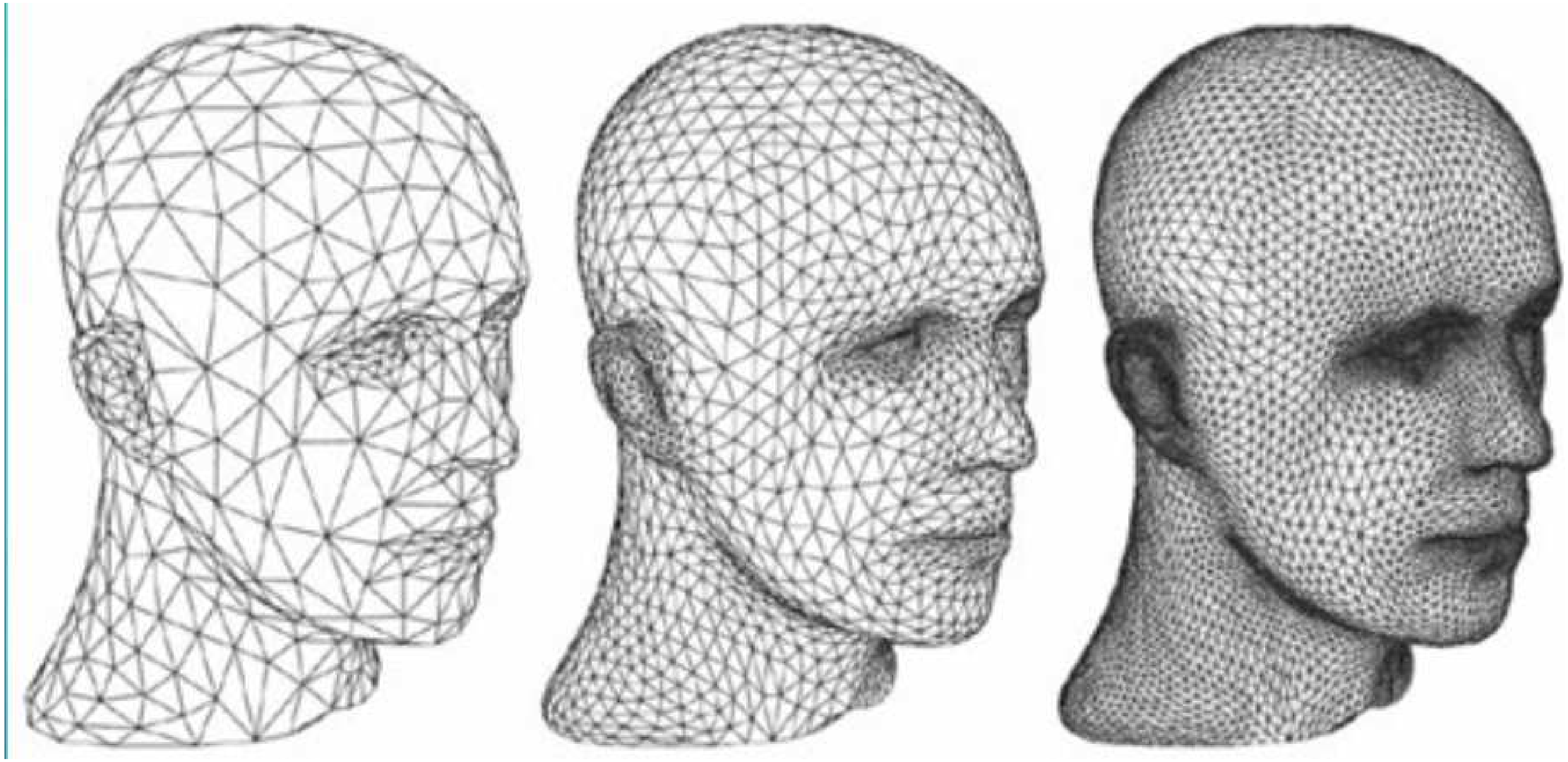


An approximate circle made of triangle primitives

Triangle Approximation of Sphere



Triangle Approximation of Human Head

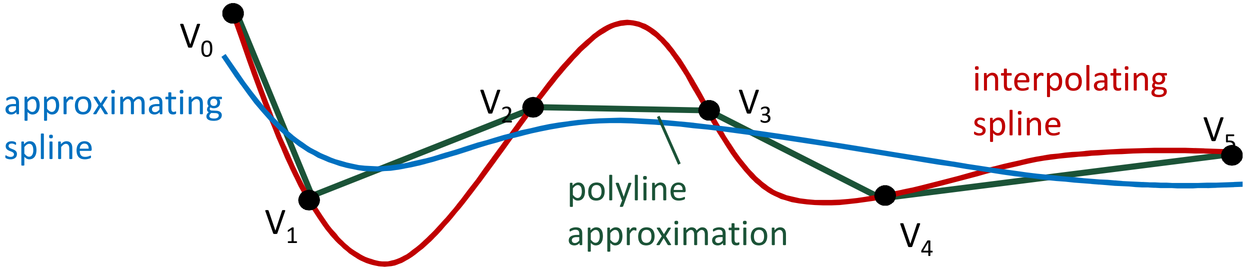


Representing Curves

- We can represent any polyline with vertices and edges. What about curves?
  - Don’t want to store curves as raster graphics (aliasing, not scalable, memory intensive). We need a more efficient mathematical representation
  - Store control points in a list, find some way of smoothly interpolating between them
  - Closely related to curve-fitting of data, done by hand with “French curves”, or by computation
- Piecewise Linear Approximation
  - Not smooth, looks awful without many control points
- Trigonometric functions (Sin(), Cos(), Tan(), etc.)
  - Difficult to manipulate and control, computationally expensive
- Higher order polynomials
  - Relatively cheap to compute, only slightly more difficult to operate on than polylines

Spline Types and Uses

- Splines: parametric curves governed by control points or control vectors, third or higher order
- Used early on in automobile and aircraft industry to achieve smoothness – even small differences can make a big difference in efficiency and look



Splines still exist outside of computers. They're now called flexible curves.

- Used for:
  - Representing smooth shapes in 2D as outlines or in 3D using "patches" parameterized with two variables: *s* and *t*
  - Animation paths for "tweening" between keyframes
  - Approximating "expensive" functions (polynomials are cheaper than log, sin, cos, etc.)

Hermite Curves

- Polylines are linear (1st order polynomial) interpolations between points
  - Given points *P* and *Q*, line between the two is given by the parametric equation:

$$x(t) = (1 - t)P + tQ, \qquad 0 \leq t \leq 1$$

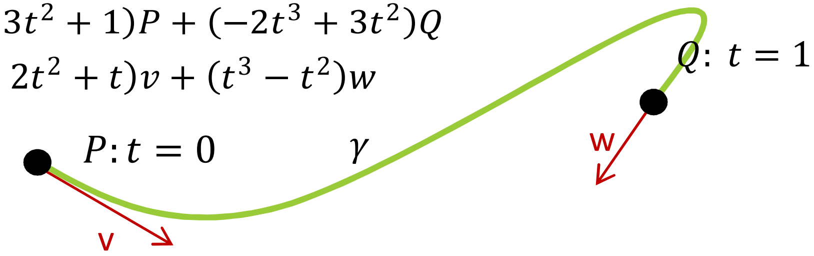
  - and t are called **weighting functions** of P and Q

- Splines are higher order polynomial interpolations between points
  - Like linear interpolation, but with higher order weighting functions allowing better approximations/smoother curves

- One representation - Hermite curves (Interpolating spline):
  - Determined by two control points P and Q, an initial tangent vector v and a final tangent vector w.

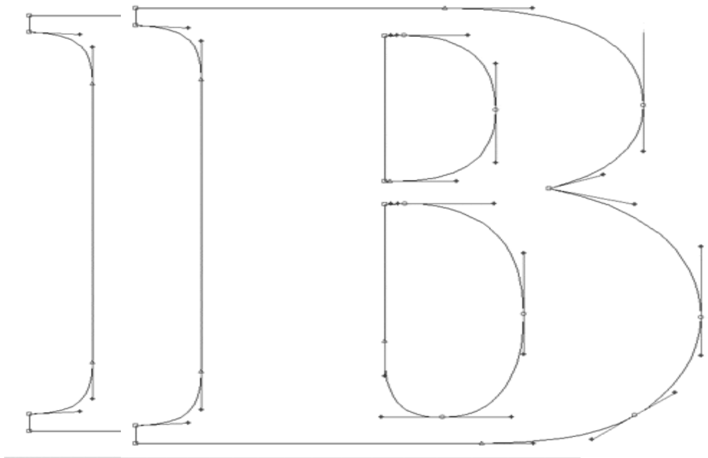
$$\gamma(t) = (2t^3 - 3t^2 + 1)P + (-2t^3 + 3t^2)Q$$
$$+(t^3 - 2t^2 + t)v + (t^3 - t^2)w$$

  - Satisfies:
    - $\gamma(0) = P$
    - $\gamma(1) = Q$
    - $\gamma'(0) = v$
    - $\gamma'(0) = v$



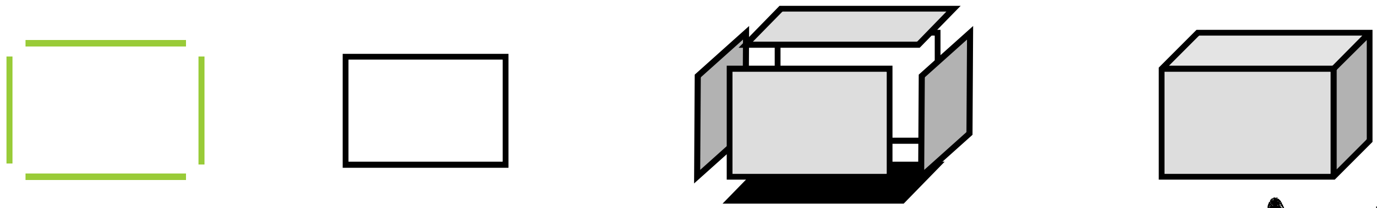
Bezier Curves

- Bezier representation is similar to Hermite
  - 4 points instead of 2 points and 2 vectors (P_1 … P_4)
  - Initial position P_1, tangent vector is $P_2 - P_1$
  - Final position P_4, tangent vector is $P_4 - P_3$
  - This representation allows a spline to be stored as a list of vertices with some global parameters that describe the smoothness and continuity
- Bezier splines are widely used (Adobe, Microsoft) for font definition
- https://www.jasondavies.com/animated-bezier/

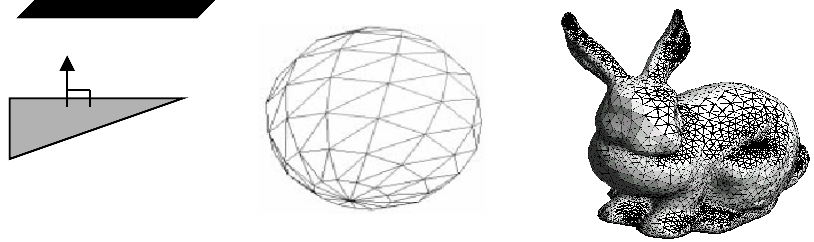


3D Primitives

- Made out of 2D and 1D primitives

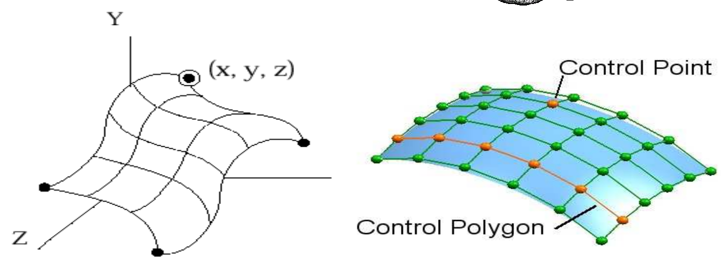


- Triangles are commonly used
- Many triangles used for a single object is a triangular mesh



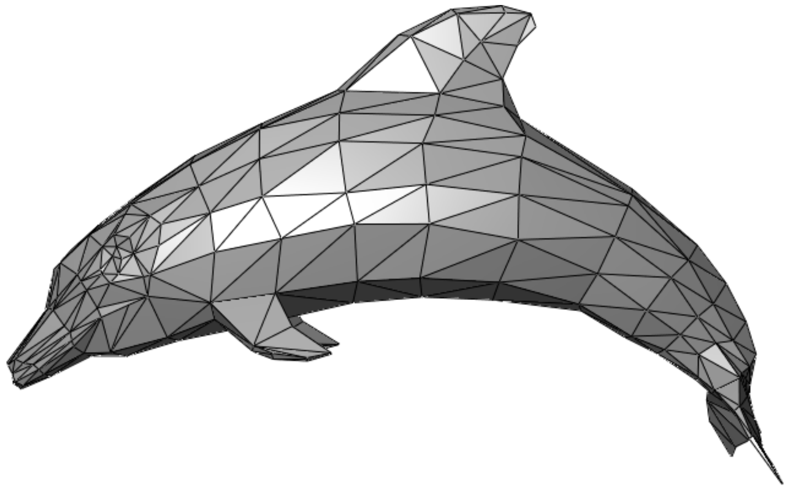
- Splines used to describe boundaries of "patches" – these can be "sewn together" to represent curved surfaces

$$x(s,t) = (1-s)^3 \times (1-t)^3 \times P_{1,1}$$
$$+(1-s)^3 \times 3t(1-t)^2 \times P_{1,2} + \cdots$$



Triangle Meshes

- Most common representation of shape in three dimensions
- All vertices of triangle are guaranteed to lie in one plane (not true for quadrilaterals or other polygons)
- Uniformity makes it easy to perform mesh operations such as subdivision, simplification, transformation etc.
- Many different ways to represent triangular meshes



Triangular Mesh Representation

- Vertex and face tables, analogous to 2D vertex and edge tables
- Each vertex listed once, triangles listed as ordered triplets of indices into the vertex table
  - Edges inferred from triangles
  - It's often useful to store associated faces with vertices (i.e. computing normals: vertex normal as average of surrounding face normals)
- Vertices listed in **counter** clockwise order in face table.
  - No longer just because of convention. CCW order differentiates front and back of face

Vertex List

v0	0, 0, 0	f0 f1 f12 f15 f7
v1	1, 0, 0	f2 f3 f13 f12 f1
v2	1, 1, 0	f4 f5 f14 f13 f3
v3	0, 1, 0	f6 f7 f15 f14 f5
v4	0, 0, 1	f6 f7 f0 f8 f11
v5	1, 0, 1	f0 f1 f2 f9 f8
v6	1, 1, 1	f2 f3 f4 f10 f9
v7	0, 1, 1	f4 f5 f6 f11 f10
v8	.5, .5, 1	f8 f9 f10 f11
v9	.5, .5, 0	f12 f13 f14 f15

Face List

f0	v0 v5 v4
f1	v0 v5 v1
f2	v1 v6 v5
f3	v1 v2 v6
f4	v2 v7 v6
f5	v2 v3 v7
f6	v3 v4 v7
f7	v3 v0 v4
f8	v8 v4 v5
f9	v8 v5 v6
f10	v8 v6 v7
f11	v8 v7 v4
f12	v9 v1 v0
f13	v9 v2 v1
f14	v9 v3 v2
f15	v9 v0 v3

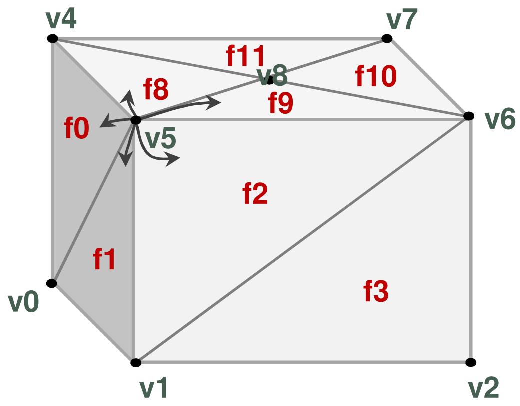


Image Processing

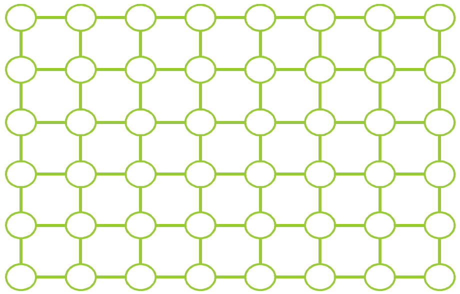
LECTURE 02: INTRODUCTION TO 2D AND 3D GRAPHICS

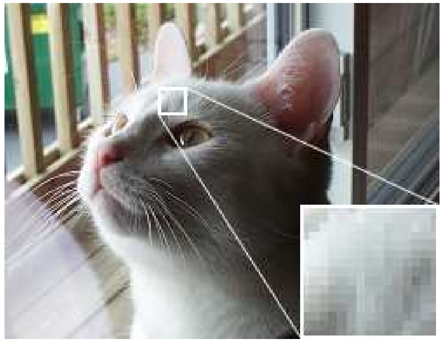
Raster Displays

- The screen is represented by a 2D array of locations called pixels
- Zooming in on an image made up of pixels

What is an image?

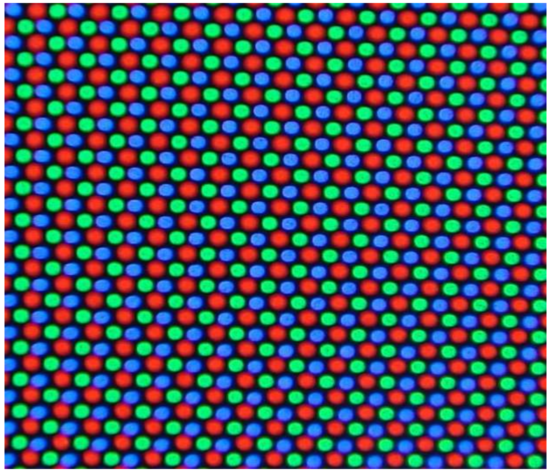
- A 2D domain with samples at regular points (almost always a rectilinear grid)
  - Can have multiple values sampled per point
  - Meaning of samples depend on the application (red, green, blue, opacity, depth, etc.)
- Units also depend on the application
  - e.g., a computed int or float to be mapped to voltage needed for display of a pixel on a screen
  - e.g., as a physical measurement of incoming light (e.g., a camera pixel sensor)



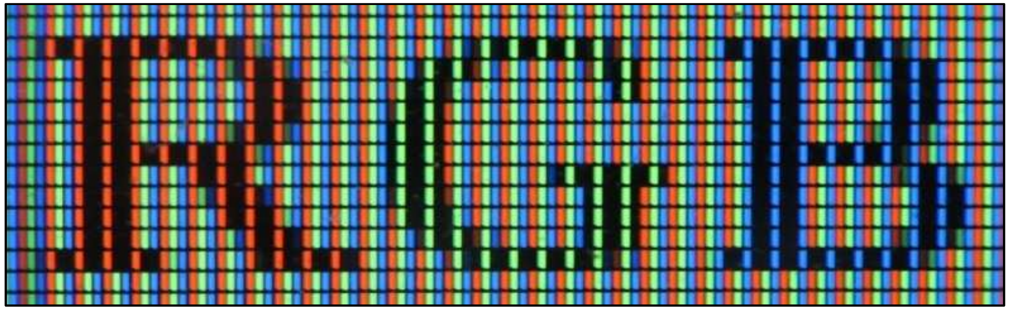


Pixels

- Pixels are **point samples**, not "squares" or "dots"
- Point samples reconstructed for display (often using multiple subpixels for primary colors)



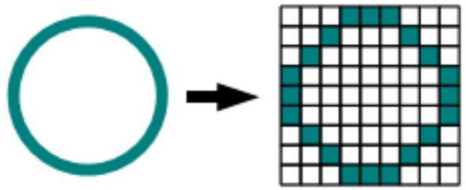
Close-up of a CRT screen

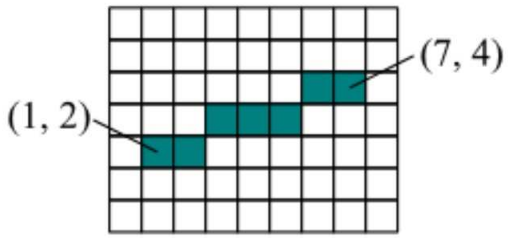


Close-up of an LCD screen

Basic Line Drawing

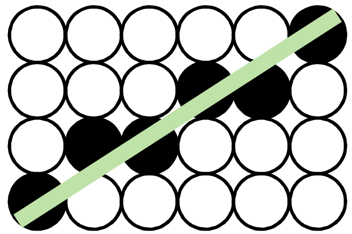
- Set the colour of pixels to approximate the appearance of a line from (x_0, y_0) to (x_1, y_1)
- It should be
  - "straight" and pass through the end points
  - Independent of point order
  - Uniformly bright, independent of slope
- The explicit equation for a line is
  - $y = mx + b$



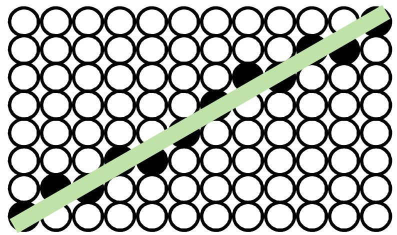


Midpoint Algorithm

- Midpoint algorithm: in each column, pick the pixel with the closest center to the line
  - A form of point sampling: sample the line at each of the integer X values
  - Pick a single pixel to represent the line's intensity, full on or full off
- Doubling resolution in x and y only lessens the problem, but costs 4 times the memory, bandwidth, and scan conversion time
  - Note: This works for -1 < slope < 1, use rows instead of columns for the other case or there will be gaps in the line

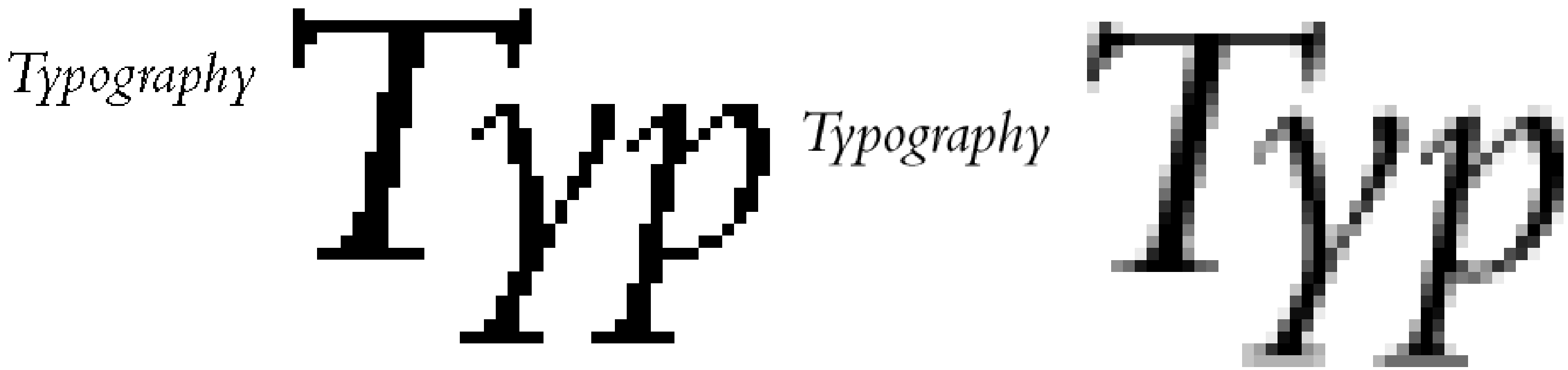


Line approximation using point sampling



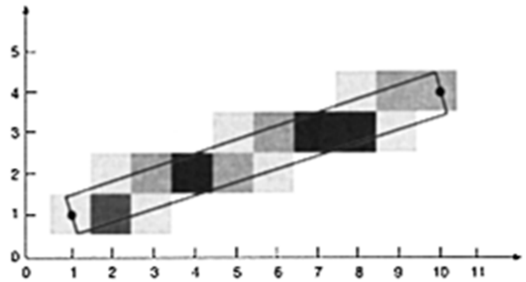
Approximating same line at 2x the resolution

Jaggies & Aliasing



Area sampling

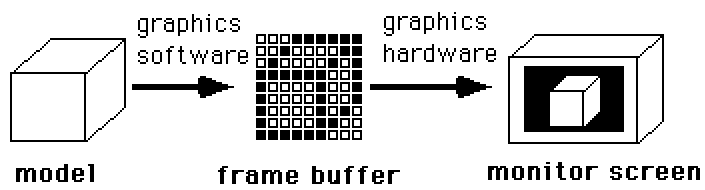
- Represent the line as a unit width rectangle, use multiple pixels overlapping the rectangle (for now we think of pixels as squares)



- Instead of full on/off, calculate each pixel intensity proportional to the area covered by the unit rectangle
- A form of unweighted area sampling – stay tuned:
  - Only pixels covered by primitive can contribute
  - Distance of pixel center to line doesn't matter
- Typically have more than one pixel per column so can go gradually from dark for pixels covered by the line to white background; the more area of overlap, the darker the pixel

Frame Buffer

- A frame buffer is characterized by size, x, y, and pixel depth
- The resolution of a frame buffer is the number of pixels in the display. e.g. Full HD 1920 x 1080 pixels
- Bit Planes or Bit Depth is the number of bits corresponding to each pixel
  - This determins thecolour resolution of the buffer



Double Buffering and Page Flipping

- Multiple frame buffers can be stored in computer memory
- Double buffering
  - First image is drawn into frame buffer and sent to display
  - While the user is looking on the display, the next picture is drawing to the second buffer
- Page flipping
  - Instead of copying the data, both butters are capable of being displayed
  - Typically accomplished by modifying the value of a point to the beginning of the dispay data in the memory