Introduction to spagetti and meatballs

CSC 418/2504: Computer Graphics

Course web site (includes course information sheet):

`http://www.dgp.toronto.edu/~karan/courses/418/`

Instructors:

L2501, T 6-8pm	L0101, W 3-5pm
Karan Singh	David Levin
BA 5258	BA 5268
978-7201	978-2052
karan@dgp.toronto.edu	diwlevin@cs.toronto.edu
office hours: T 5-6pm	*office hours:* W 2-3pm
or by appointment.	or by appointment.

Textbooks: Fundamentals of Computer Graphics
OpenGL Programming Guide & Reference

Tutorials: (first tutorial next week)


This is a Green Course!
UofT
IT'S GREENER
HERE
Your instructor has committed to reducing this course's environmental impact.
sustainability.utoronto.ca
#greenerUofT

Topics

0. Introduction: What is Computer Graphics?
1. Basics of scan conversion (line drawing)
2. Representing 2D curves
3. 2D Transformations
4. Coordinate Free Geometry CFG
5. 3D Objects
6. 3D curve design
7. 3D Transformations
8. 3D Viewing
9. Visibility
10. Lighting and local illumination
11. Shading
12. Texture mapping
13. Ray Tracing and Global illumination
14. Animation

Topic 0.

Introduction:
What Is Computer Graphics?

What is Computer Graphics?

Computers:

accept, process, transform and present information.

Computer Graphics:

accept, process, transform and present information in a visual form.

Ok but… what is the course really about?

The science of turning the rules of geometry, motion and physics into (digital) pictures that mean something to people

What its not about?

Photoshop, AutoCAD, Maya, Renderman, Graphics APIs.

…**wow**, heavy math and computer science!!

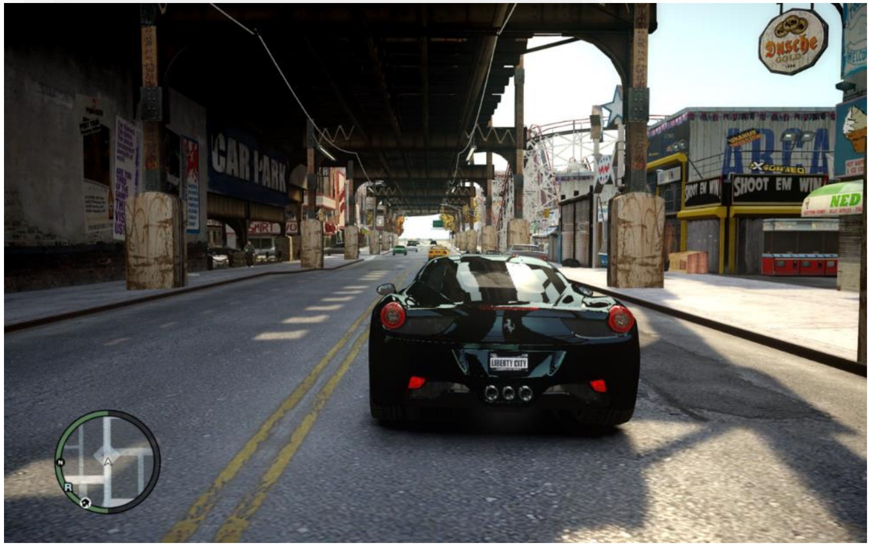
Movies

Movies define directions in CG
Set quality standards
Driving medium for CG

Games

Games emphasize the interactivity and AI

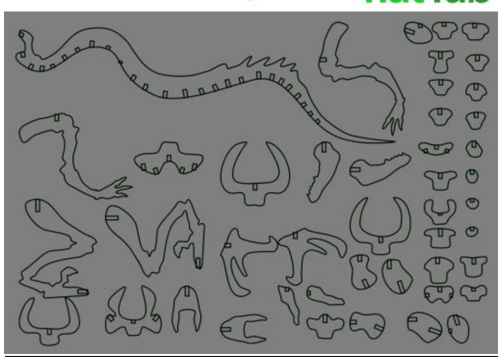
Push CG hardware to the limits (for real time performance)

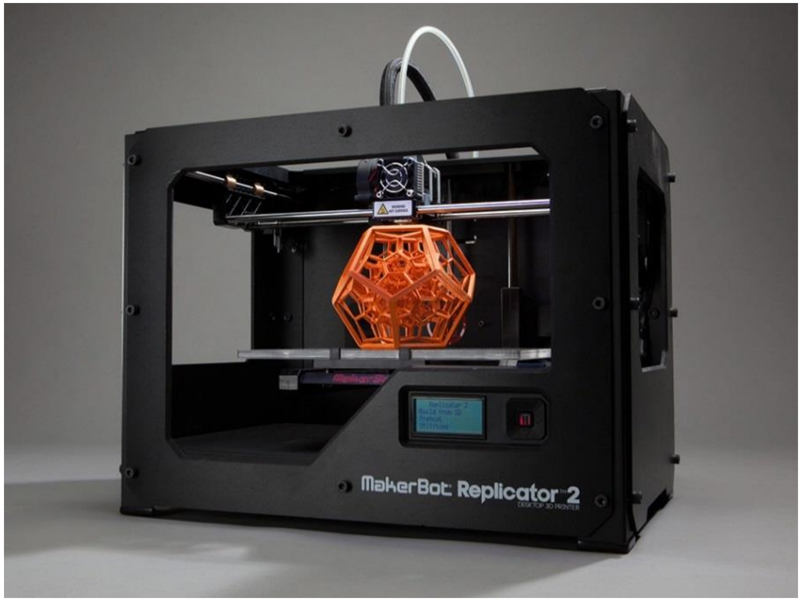


Design

CG for prototyping and fabrication

Requires precision modeling and engineering visualization



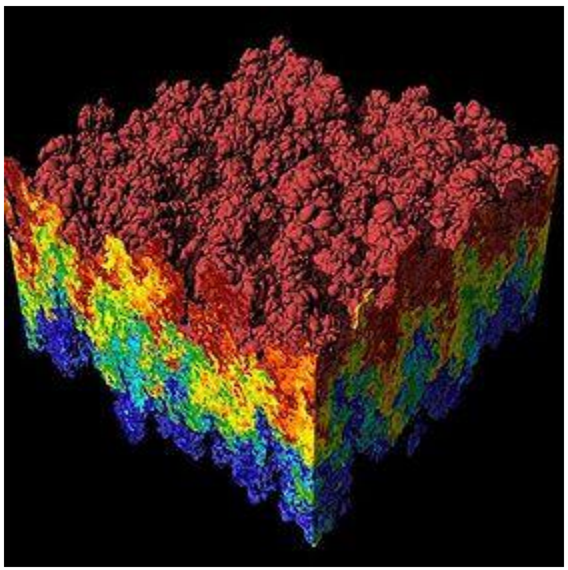


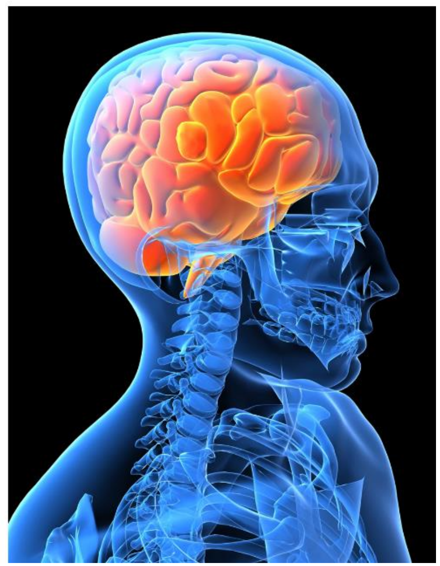
Scientific and Medical Visualization, Operation

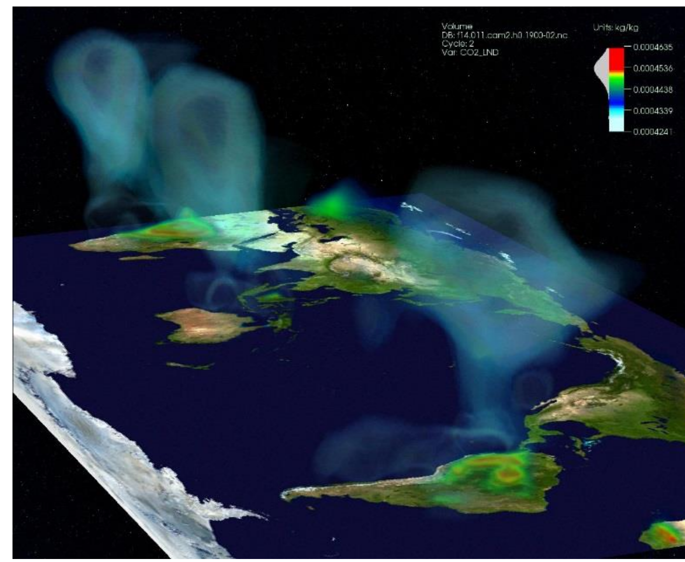
Requires handling large datasets

May need device integration

Real-time interactive modeling & visualization



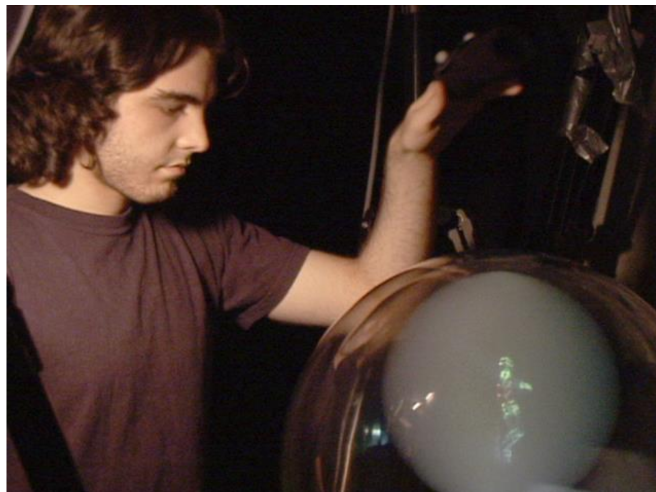


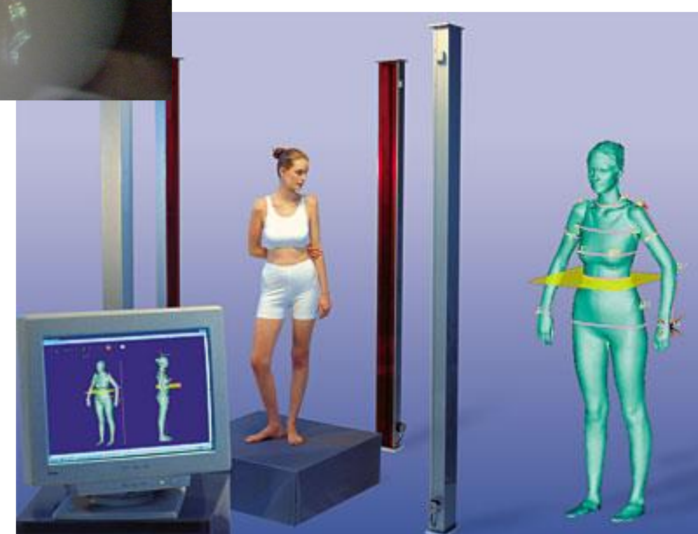


GUIs, AR/VR, scanners…

Interaction with software & hardware, I/O of 3D data

Emphasis on usability





Computer Graphics: Basic Questions

- Form (modeling)

  How do we represent (2D or 3D) objects & environments?

  How do we build these representations?

- Function, Behavior (animation)

  How do we represent the way objects move?

  How do we define & control their motion?

- Appearance (rendering)

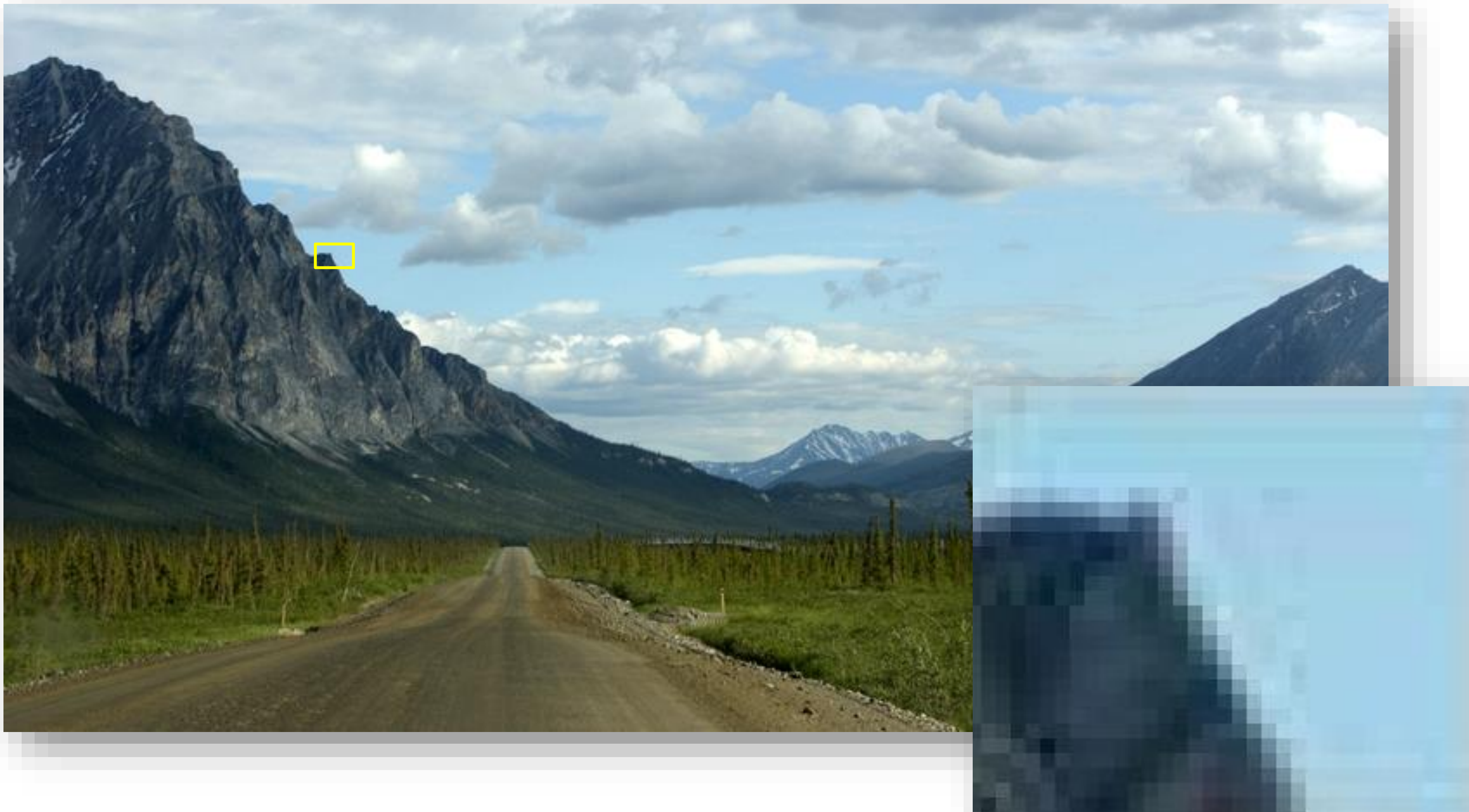
  How do we represent the appearance of objects?

  How do we simulate the image-forming process?

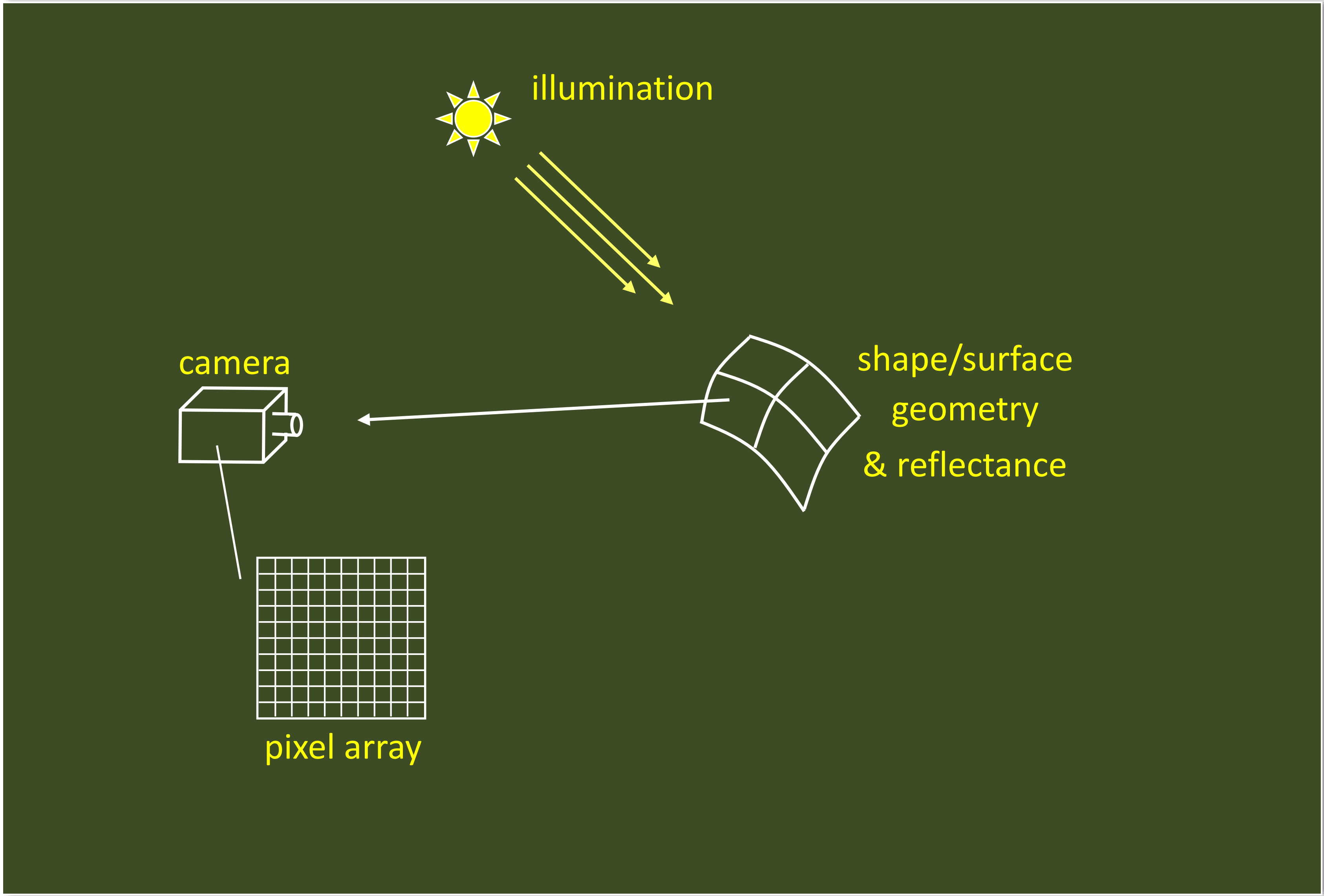
What is an Image?

Image = distribution of light energy on 2D "film"

Digital images represented as rectangular arrays of <u>pixels</u>



Form & Appearance in CG



The Graphics Pipeline

Modeling → Animation → Rendering

Modeling

- Geometry: points, curves, & surfaces
- Scene Objects: parts, relations, & pose
- Texture and reflectance (e.g., color, diffusivity, opacity, refractions)
- …

Animation

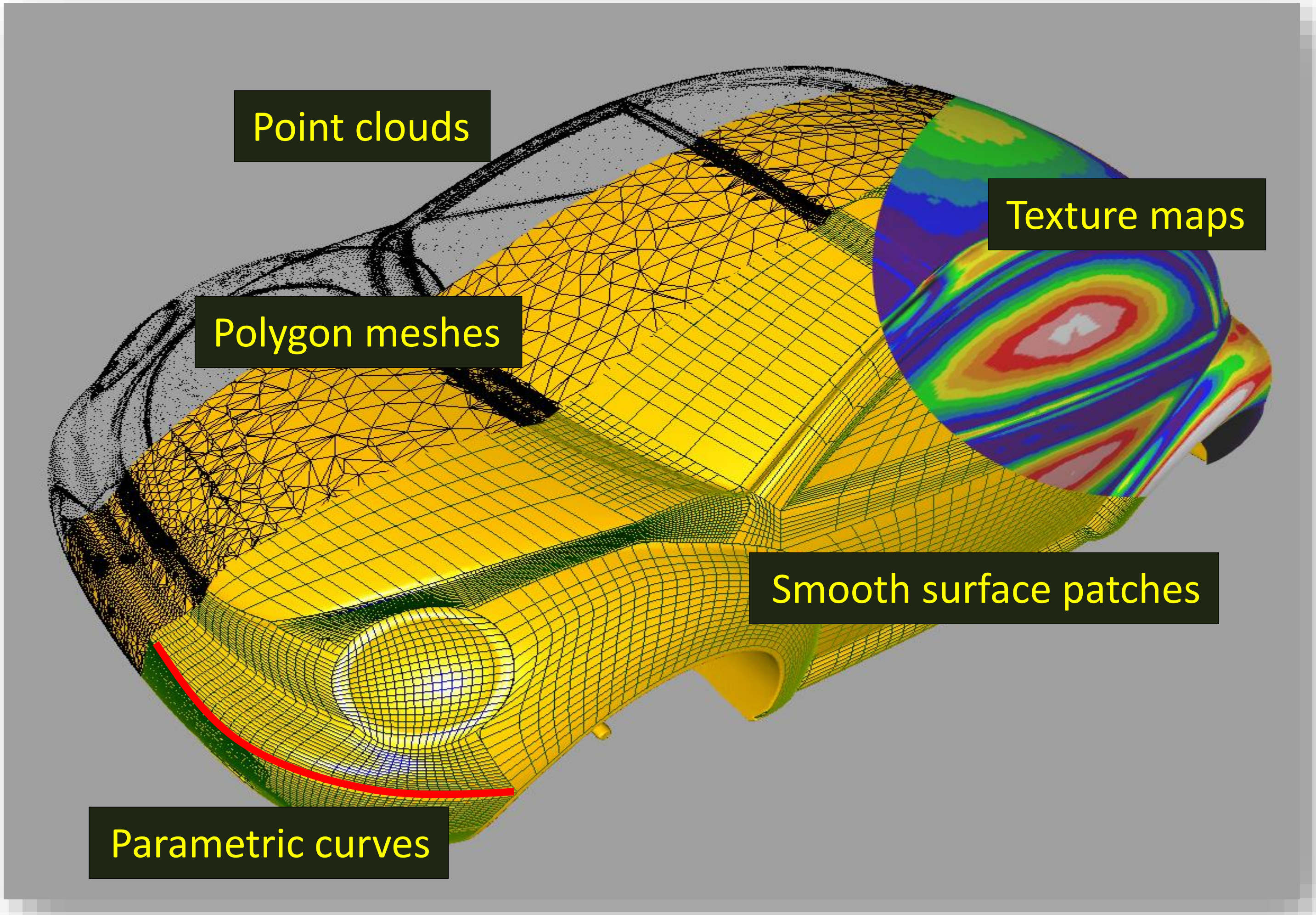
- Key-frame, motion capture, inverse kinematics, dynamics, behaviors, motion planning, …

Rendering

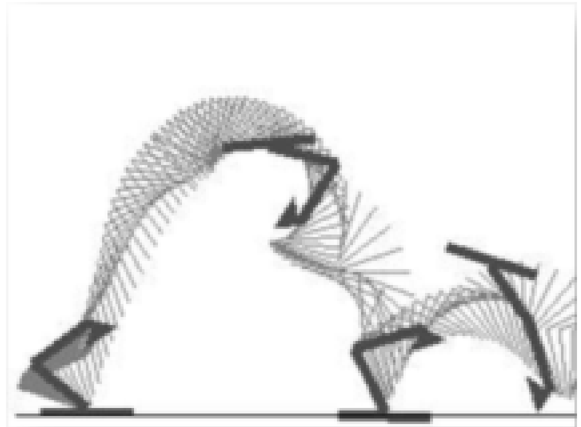
- Visibility
- Simulation of light (e.g., illuminants, emissive surfaces, scattering, transmission, diffraction, …)
- Special effects (e.g., anti-aliasing, motion blur, non-photorealism)

Graphics Pipeline: Modeling

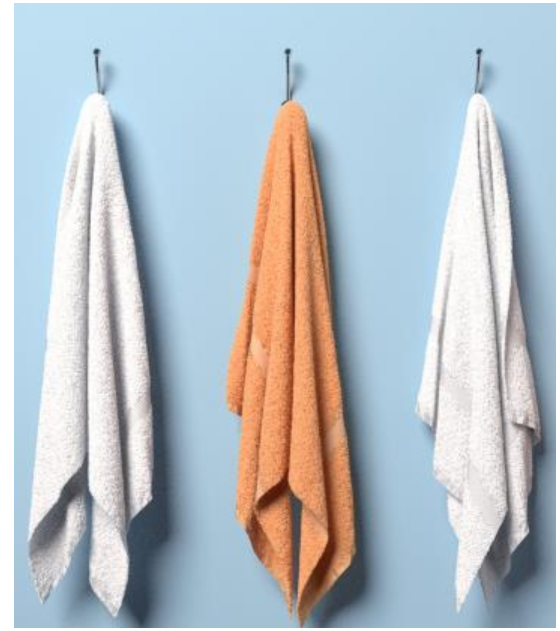
How do we represent an object geometrically on a computer?



Graphics Pipeline: Animation



Physical simulation



Key-Framing

Behavior rules

Graphics Pipeline: Rendering

Input: Scene description, lighting, camera

Output: Image that the camera will observe…
accounting for visibility, clipping, projection,…

Course Topics

Principles

Theoretical & practical foundations of CG
(core mathematics, physics, modeling methods)

CG programming (assignments & tutorials)

- Experience with OpenGL (industry-standard CG library)
- Creating CG scenes

What You Will Take Away …

#1: yes, math <u>IS</u> useful in CS !!

#2: how to turn math & physics into pictures.

#3: basics of image synthesis

#4: how to code CG tools

Administrivia

Grading:

- 50%: 3 assignments handed out in class
  (25% 15% 10%).
- 50%: 1 test in class (15%) + 1 final exam (35%).
- First assignment: on web in two weeks.
- Wooden Monkey assignment on web now!
- Check web for schedule, dates, more details & policy on late assignments.

Tutorial sessions:

- Math refreshers, tutorials on OpenGL and other graphical libraries, additional topics.
- Attendance STRONGLY encouraged since I will not be lecturing on these topics in class.

Lecture slides & course notes, already on web.

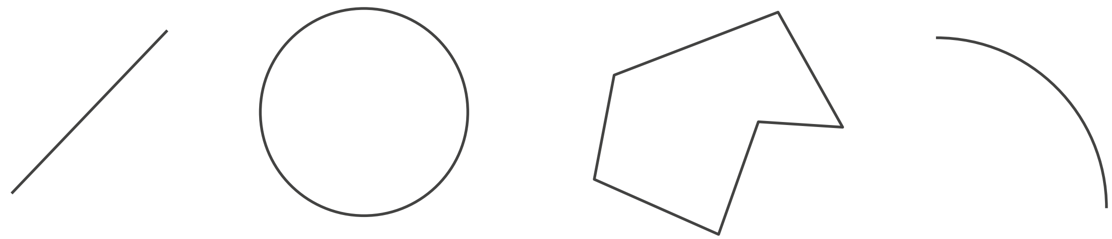
Topic 1.

Basic Raster Operations: Line Drawing

- A simple line drawing algorithm
- Line anti-aliasing

2D Drawing

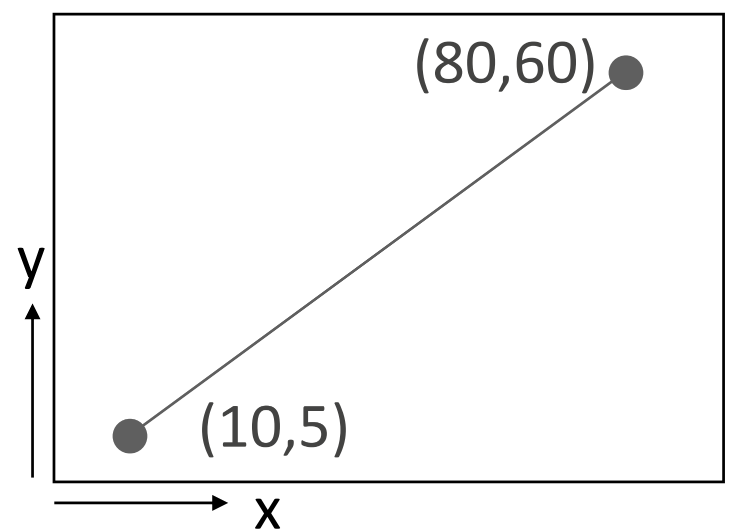
Common geometric primitives:



When drawing a picture, 2D geometric primitives are specified as if they are drawn on a continuous plane

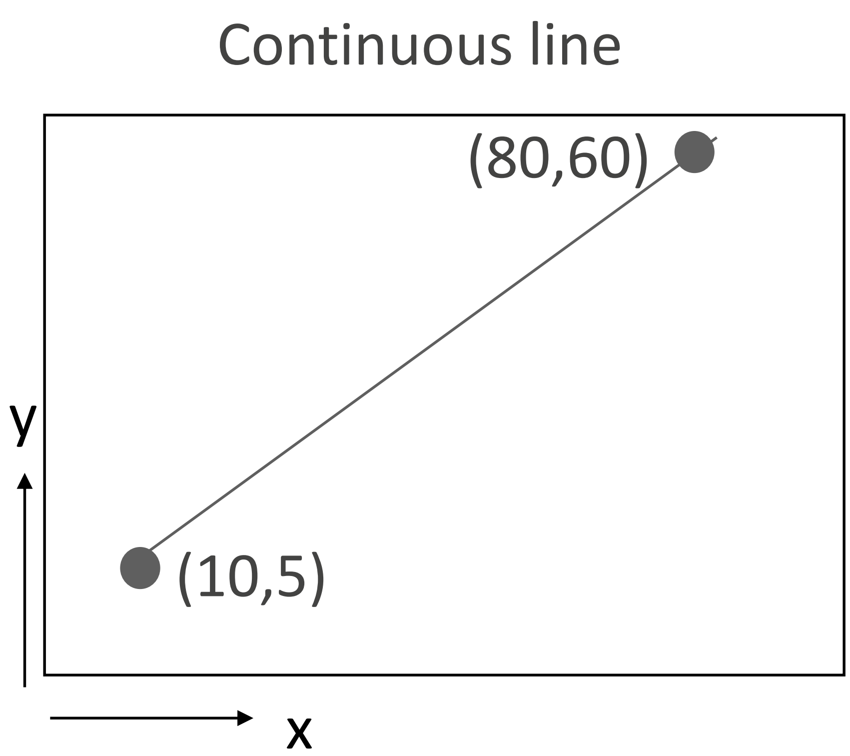
Drawing command:

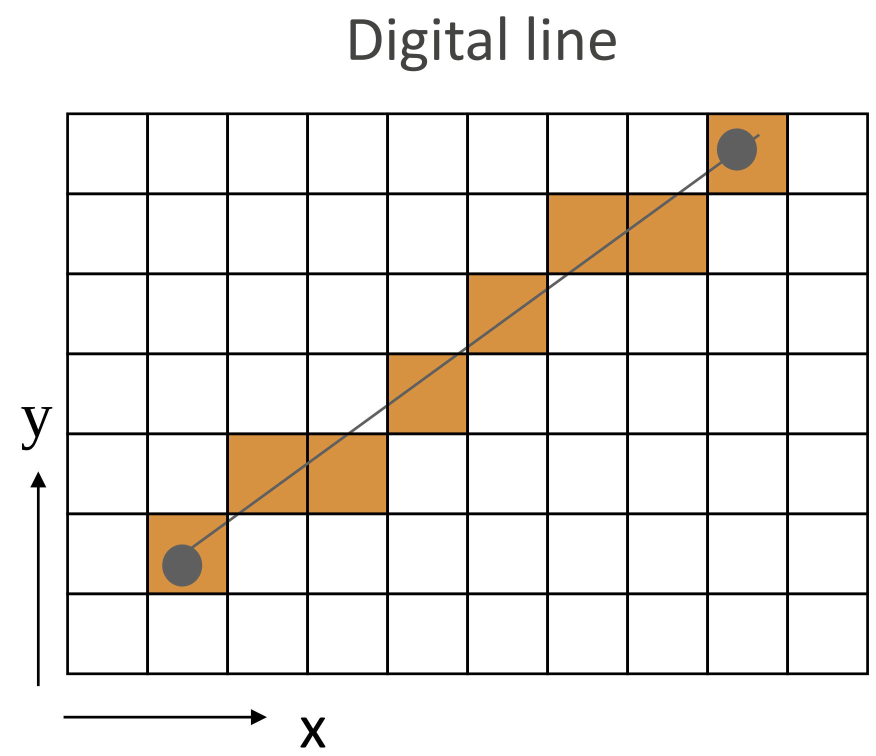
Draw a line from point (10,5) to point (80,60)



2D Drawing

In reality, computer displays are arrays of pixels, not abstract mathematical continuous planes





In graphics, the conversion from continuous to discrete 2D primitives is called scan conversion or rasterization

Basic Raster Operations (for 2D lines)

- Scan conversion: Given a pair of pixels defining the line's endpoints & a color, paint all pixels that lie on the line.
- Clipping: If one or more endpoints is out of bounds, paint only the line segment that is within bounds.
- Region filling: Fill in all pixels within a given closed connected boundary of pixels.

Line Scan Conversion: Key Objectives

Accuracy:

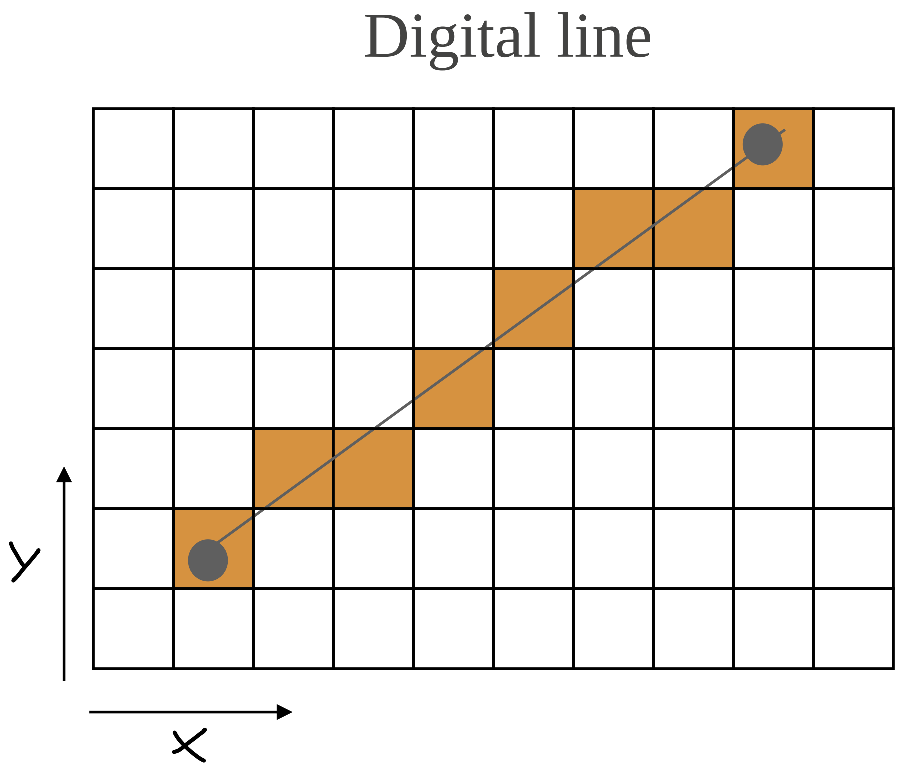
pixels should approximate line closely.

Speed:

line drawing should be efficient

Visual Quality:

No discernable “artifacts”.



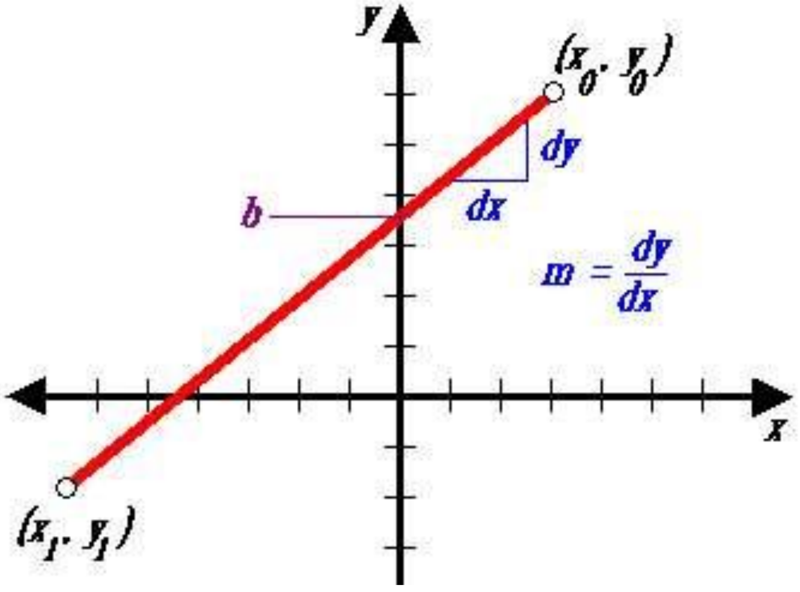
Equation of a Line

Line between (x_0,y_0) and (x_1,y_1)

$dx = x_1 - x_0$, $dy = y_1 - y_0$

Explicit : $y = mx + b$

$m = dy/dx$, $b = y_0 - mx_0$



Parametric :

$x(t) = x_0 + dx*t$

$y(t) = y_0 + dy*t$

$P = P_0 + (P_1 - P_0)*t$

$P = P_0*(1-t) + P_1*t$ (weighted sum)

Implicit : $(x-x_0)dy - (y-y_0)dx = 0$

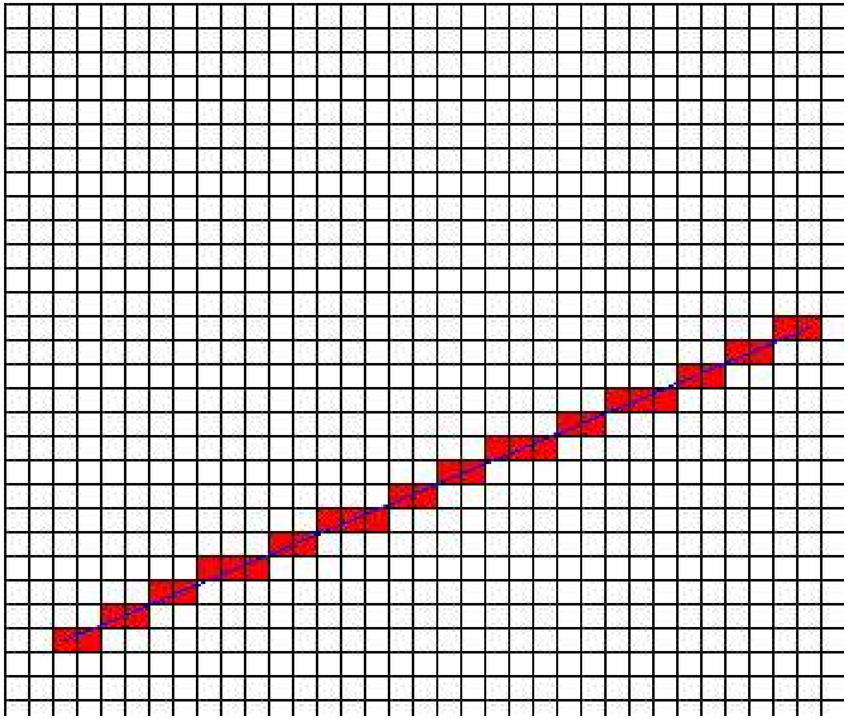
Algorithm I

DDA (Digital Differential Analyzer)

Explicit form:

y= dy/dx * (x-x0) + y0

```
float y;
int x;
dx = x1-x0; dy = y1 – y0;
m = dy/dx;
y= y0;
for ( x=x0; x<=x1; x++)
{
   setpixel (x, round(y));
    y= y + m;
 }
```

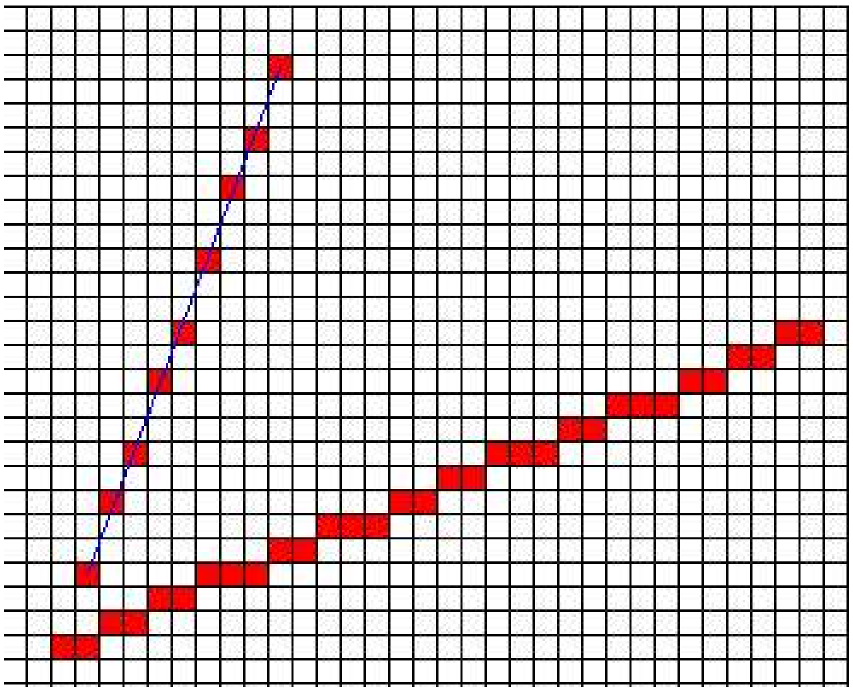


Algorithm I (gaps when m>1)

DDA (Digital Differential Analyzer)

Explicit form:

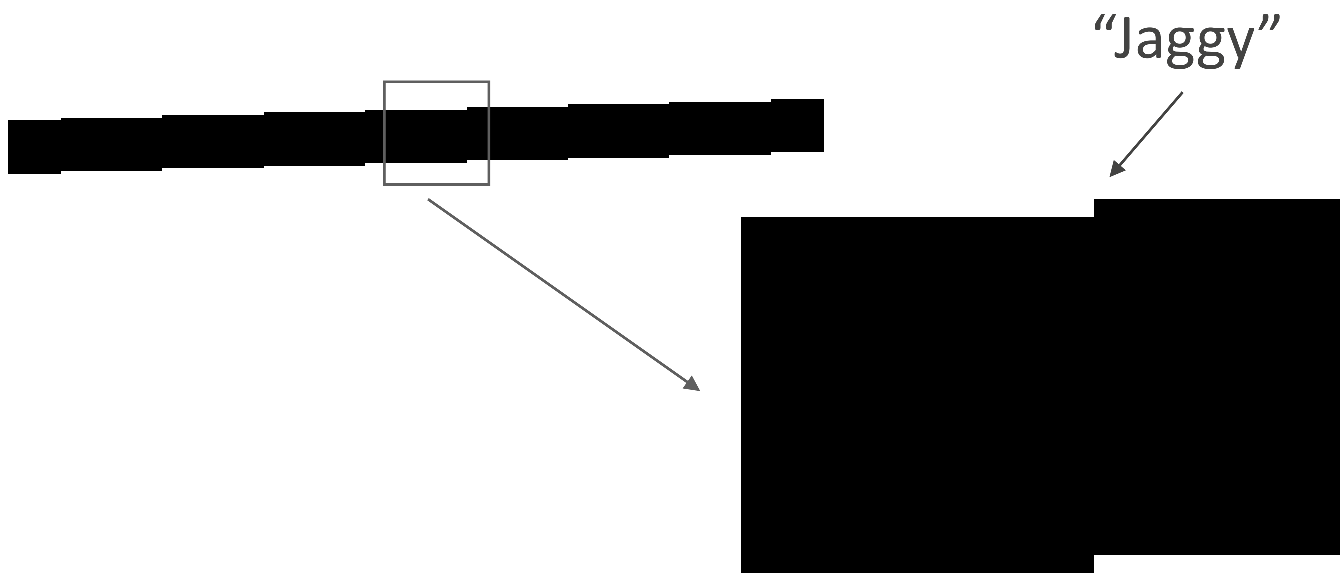
y= dy/dx * (x-x0) + y0



```
float y;
int x;
dx = x1-x0; dy = y1 – y0;
m = dy/dx;
y= y0;
for ( x=x0; x<=x1; x++)
{
   setpixel (x, round(y));
    y= y + m;
 }
```

Aliasing

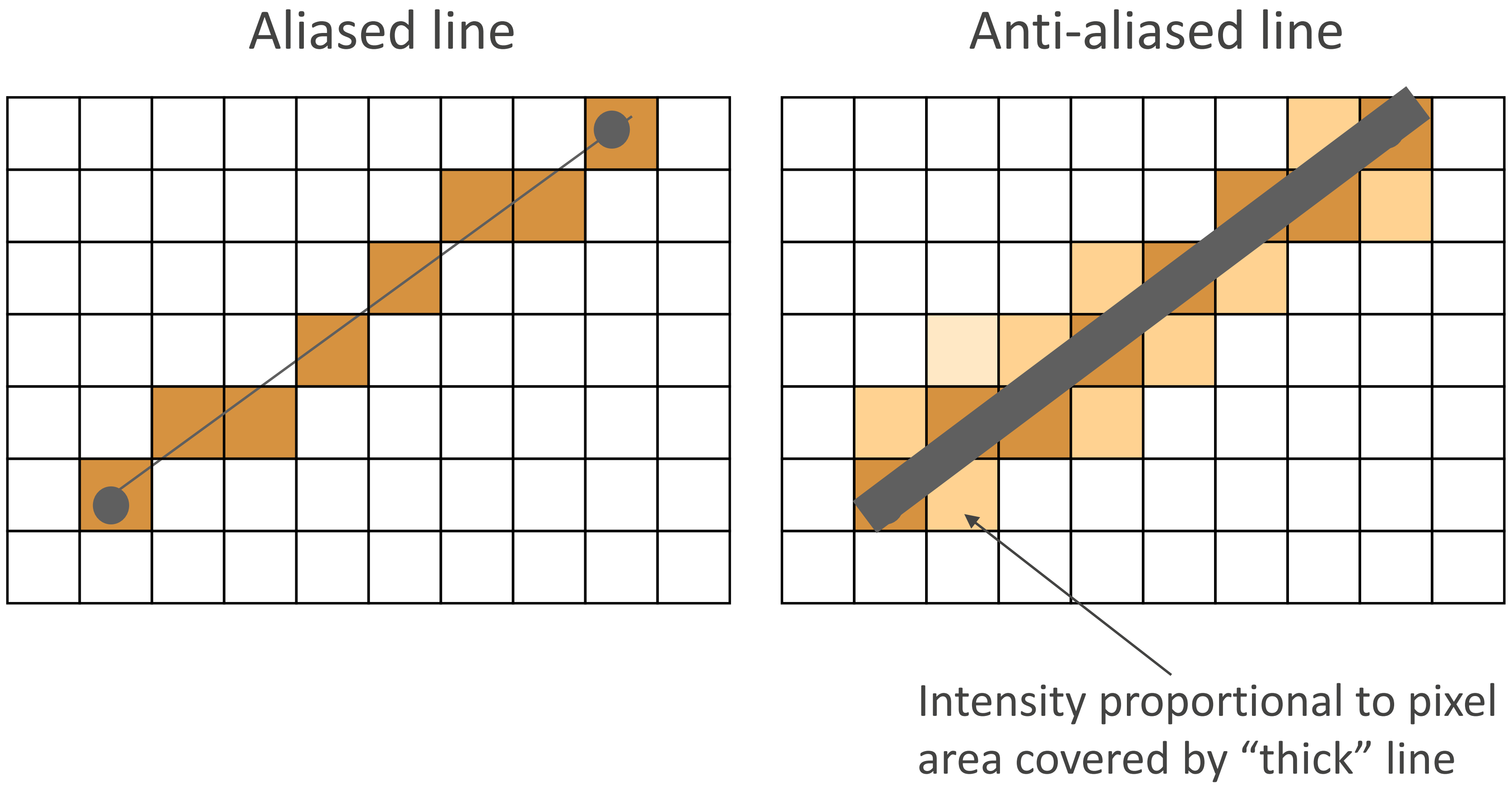
Raster line drawing can produce a “jaggy” appearance.



- Jaggies are an instance of a phenomenon called aliasing.
- Removal of these artifacts is called anti-aliasing.

Anti-Aliasing

How can we make a digital line appear less jaggy?



Main idea: Rather than just drawing in 0's and 1's, use "in-between" values in neighborhood of the mathematical line.

Anti-Aliasing: Example

Topic 2.

2D Curve Representations

- Explicit representation
- Parametric representation
- Implicit representation
- Tangent & normal vectors

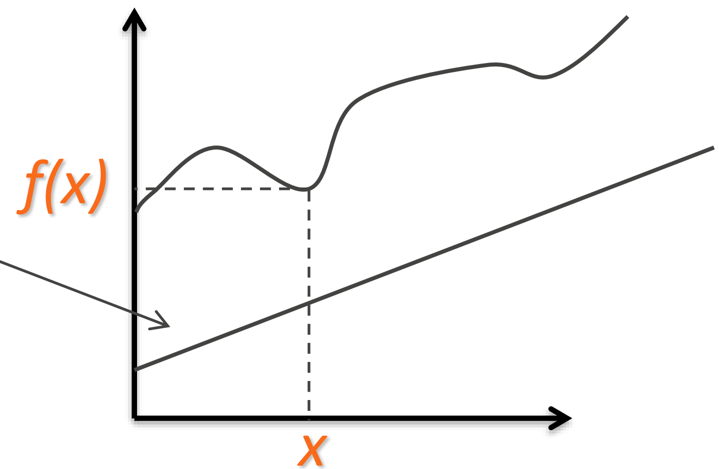
Explicit Curve Representations: Definition

Curve represented by a function *f*

such that:

y=f(x)

line: y=mx+b

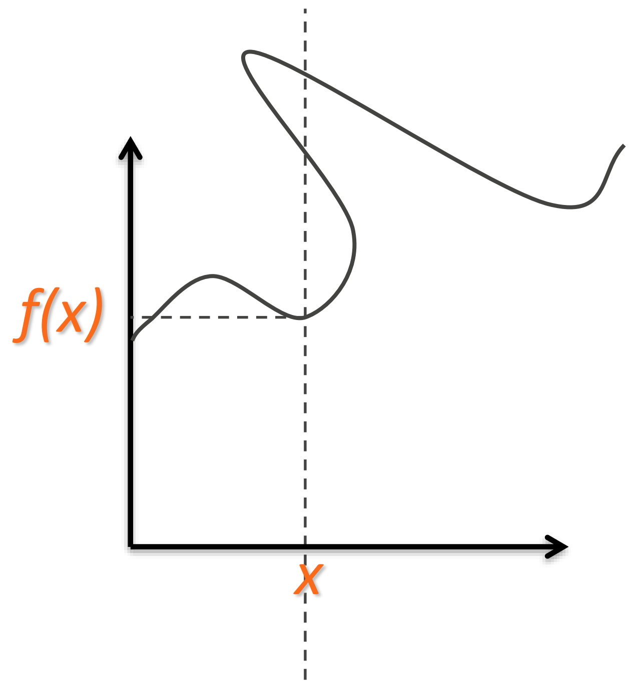


Explicit Curve Representations: Limitations

Curve represented by a function *f*

such that:

$$y=f(x)$$



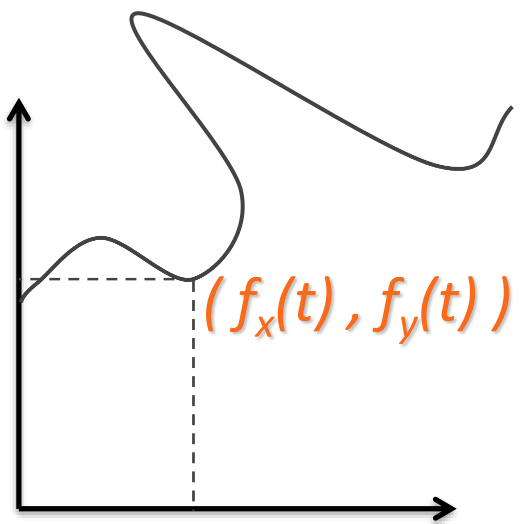
Parametric Curve Representation: Definition

Curve represented by two functions *f_x , f_y*

And an interval *[a,b]*

such that:

$$(x,y)=(f_x(t) , f_y(t))$$



are points on the curve for

t in [a,b]

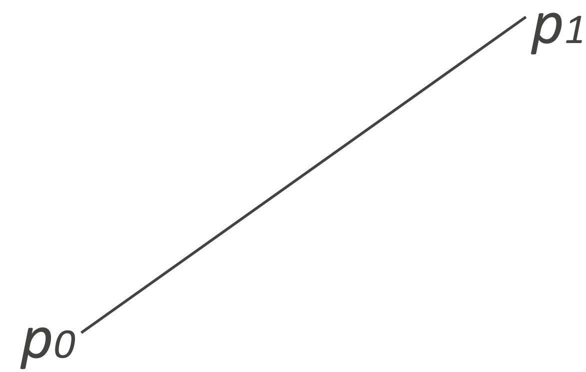
A curve is closed when ??

Parametric Representation of a Line Segment

$pt) = p_0 + (p_1 - p_0)*t \;,\; 0 \le t \le 1$

$0 \le t \le \infty$: ray from p_0 through p_1

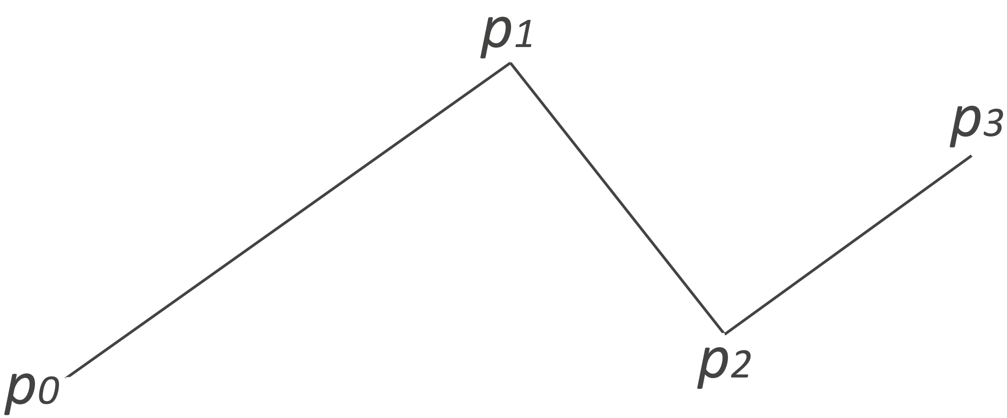
$-\infty \le t \le \infty$: line through p_0 and p_1



In general if $p(t) = a_0 + a_1*t$, how do you solve for a_0, a_1 ?

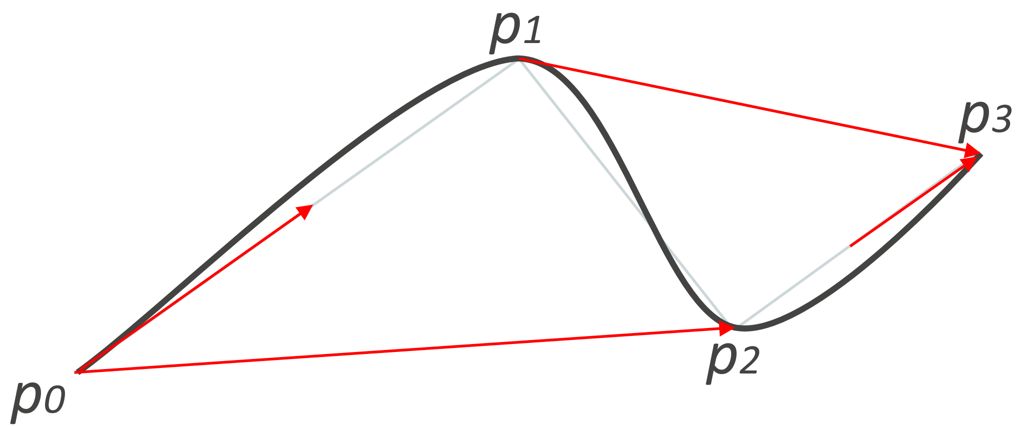
Line Segment as interpolation

$p(t) = a_0 + a_1 * t$



Curve as interpolation (Catmull-Romm)

$p(t) = a_0 + a_1 * t + a_2 * t^2 + a_3 * t^3$

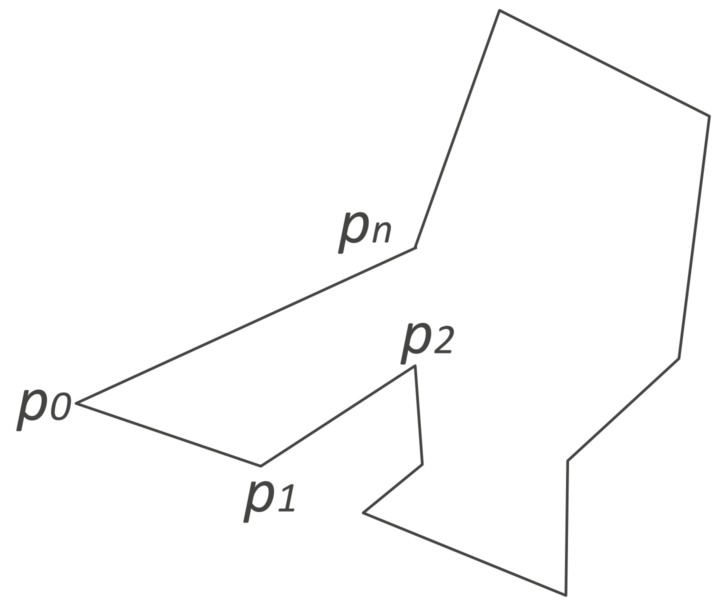


Polygons

Polygon: A continuous piecewise linear closed curve.

Simple polygon: non-self intersecting.
Convex: all angle less than 180 degrees.
Regular: simple, equilateral, equiangular.

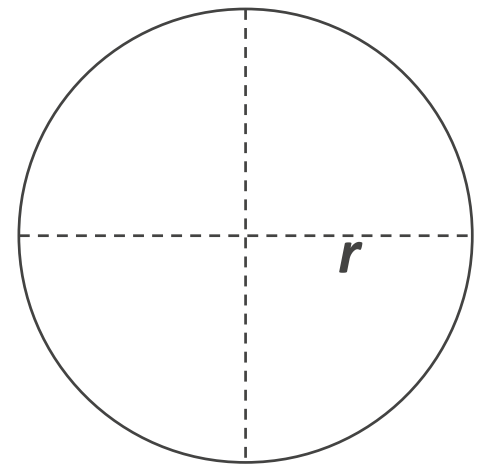
n-gon: $p_i = r(\cos(2\pi i/n), \sin(2\pi i/n))$, $0 \leq i < n$



Representations of a Circle

Parametric:

$p(t) = r(\cos(2\pi t), \sin(2\pi t))\ , \ \ 0 \le t \ \le 1$



Implicit:

$x^2+y^2-r^2=0$

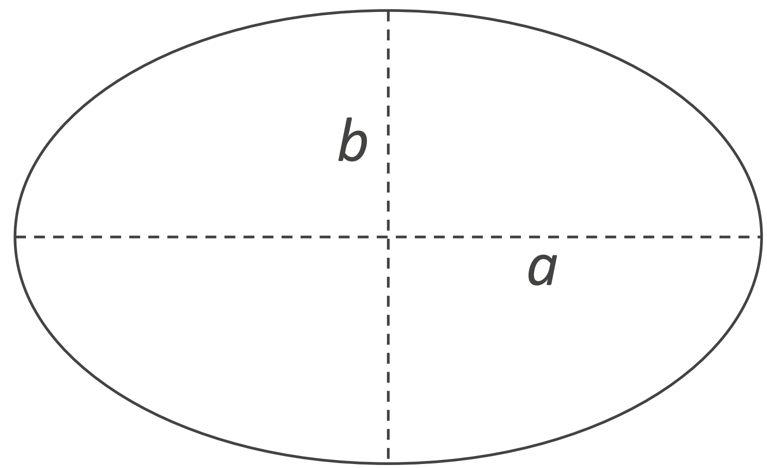
Representations of an Ellipse

Parametric:

$p(t) = (a*\cos(2\pi t), b*\sin(2\pi t)),\ 0 \leq t \leq 1$

Implicit:

$x^2/a^2+y^2/b^2-1=0$



Curve tangent and normal

Parametric:

p(t) = (x(t),y(t)). Tangent: ***(x'(t),y'(t)).***

Implicit:

f(x,y) =0. Normal: ***gradient(f(x,y)).***

Tangent and normal are orthogonal.