# cs430 9-28-2017

Stephen Slatky

## Contents



# 1 Intro

TA: Alex duff

Provides fundamentals of 2d and 3d graphics * Representations (lines, curves, polygons) * Drawing, clipping, transformations, viewing * Implementations of basic graphics systems.

Assignments will build upon each other.

You can use any lang.

7 Programming assignments

Drops lowest assignment

1 point off for every hour late

### 1.1 Topics

- MATH
- Primitive reps of simple figures.
- Drawing algorithms
- Clipping algorithms
- 2d/3d viewing
- Hidden surface removal.
- color and lighting (wrap up)

## 2 Lecture

What we want to accomplish is a software service pipeline

### 2.1 Raster Image

Specify objects via geometry and then produce a 2D image

Fast

But vector allows for better scaling

### 2.2 Pipeline

How something goes from privative to graphics on screen.

Vertices -> vertex processor -> Clipper and primitive -> Rasterize -> Fragment Processor -> Pixels

### 2.3 Math you will need

#### 2.3.1 Geometry

##### 2.3.1.1 Affine

Scalers ,points, vectors, and their operators

##### 2.3.1.2 Euclidean

- affine lacks angles and distance
- inner/dot product * Length Distance normalization

#### 2.3.1.3 Projector

- How to display
- warping?

## 2.4 Points

Points will be the base for everything we use

## 2.5 Vectors

- Directed line segment
- Magnitude

**Note**: No actual position in space

Can be defined by two points via subtraction

v = P - Q

This means: Vector from Q to P

### 2.5.1 Arrive at a new point

v = P + Q

### 2.5.2 Operations

Every vector has an inverse

Can be multiplied by a scaler

Sum of two vectors is a vector

### 2.5.3 Parametric form of lines

Given an initial point and a vector we can find any point. P(alpha) on the line alpha units from P_0 in the direction of d.

P(alpha) = P_0 + alpha(d)

## 2.6 Dot Product

u . v = N Sum k=1 u_kv_k = u_1v_1+ u_2v_2 + …

To compute the distance from the origin to a point.

Euclidean distance from (x,y) to (0,0) Sqrtx^2 + y^2)

Or the n direction continue the formula.

sqrt(p_1^2+ p_2^2 + … + P_n^2)

## 2.7 Unit Vector

Has length 1. |v| Also called normalizing.

### 2.7.1 Helpful for angles

`u . v = |u||v|cos(angle)`

### 2.7.2 Projection

We can use this to project one vector onto another. we can project u onto v as u_1 as cos(angle) v where angle is the angle between u and v.

## 2.8 Matrix Operations

### 2.8.1 Determinants

- single real number
- Computed recursively * for a fixed row i

det(A) = n Sum j= 1 A_i_j(-1)^(i+j)(M_i_J)

### 2.8.2 Cross product

Given two non-parallel vectors, a and b a X b calculates a third vector n that is orthogonal to both and b

Right hand rule - explains the direction. of the normal vector n = aXb

axb is not bXa

It is the thing with crossing one direction minus cross in other direction This is a not in depth explainion

#### 2.8.2.1 Example

Look at powerpint online

### 2.8.3 Matrix Transpose and inverse

facts * $(A^{t)}t$ = A * (A + B)^t = A^t **Fill in rest from slide**

# 3 Second half of lecture

## 3.1 Motivation

User specifies the shape type

**In this lecture** We will talk about how to create an object with endpoints

## 3.2 Scan-Conversion Algorithms

compute pixel coords for ideal line on 2D raster grid

## 3.3 Drawing a line

given two end points

we can compute the slope with simple math

dealta(y)/delta(x) = m

It will not form a perfect line with pixels. We need to start making assumtions of which pixel to fill in.

If we want to draw a line we can:

**Ineffecient way**

- compute the next x position
- then compute the next y going back to the slope formula * may not be in a strict pixel location so we need to round to an interger location

**Ineffecient way**

Less mutiplication Makes m = delta(y)/1 x_(i+1) = x_i + 1 y_(1+1) = y_i + m

### 3.3.1 Digital Differential Analyzer (DDA)

if |m| < 1 * Deltax = 1 and delta(y) = m * Start left must point * End right most point else * Delta(y) = 1 and delta(x) = 1/m * start at bottom most point * end top most point

check for vertical line case m = inf

With this we still have floating point operations

### 3.3.2 Bresenham's algorithm

No floating point operations

Based on implicit equation of a line: ( , ) = + + = 0

Zoned out for this part. Look at powerpoint

= = 2 − 2 + (2 − + 2( − x))

** 5 Diffderent cases ** 1. Vertical line 2. 0 <= m <= 1 * if q> r swap points 3. m > 1 4. -1 <= m < 0 * if q> r swap points 5. m < -1

#### 3.3.2.1 Example

m < -1 (Px,Py) (Px + 1/2, Py-1)

# 4 Assigiment details

- read in post script. (.ps)
- Simulate writing to a frame buffer by creating XPM image
- Draw lines

** It will all build off of each other so make it modular when designing.**

- All black and white.
- Create data structes to hold points

# 5 Glossary

**Orthogonal** - Perpendicular