CS 5400

Manipulation of pictorial data by a computer

Graphics system components

CPU

GPU

Has its own memory

Memory

Framebuffer

Chunk of memory that is in the GPU

Output devices

Cathode ray tube

LCD liquid crystal display

White light shines through 2 polarizers and liquid crystal to get color

OLED

Pixels themselves emit the light (RGB)

Pixels and framebuffer

Modern graphics – raster based

An array of pixels

Entire array is called the framebuffer

Depth or precision

Bits used to represent each pixel

How many colors are represented

True color / full-color

Has 24 bits per pixel, 8 for red, 8 for green, 8 for blue

High dynamic range (HDR)

Ratio between max and min intensifies in image/display

HDR has 10 bits per color (1024 shades of each color including black)

Dynamic range of 12 bits per color is 4096:1

Human eye is estimated to have dynamic range of 10,000:1

4K (ultra hd) 2840 x 2180 = 8.2 million pixels

At 24 bits per pixel that is 25 MB

36 bits per pixel is 37 MB of memory

Coordinates

Framebuffer is drawn to an output device

Device coordinates = screen coordinates

Upper left is (0, 0) bottom right is the resolution (max x and y)

World coordinates

Device independent representation of data before it is rasterized for display

X and y between -1 and 1 and converted to device coordinates upon display

World to device

X = deviceSizex / 2 + wordX \* deviceSizeX / 2

Device to world

WorldX = (DeviceX – DeviceSizeX / 2) / (DeviceSizeX / 2)

GPU

Started as a fixed function (built-in functions)

High-degree of parallelism, thousands of vector processing units

Thousands of instructions per clock-cycle

SIMD vector and matrix operations (single instruction for matrix multiply)

Dedicated memory including the framebuffers from which the display is generated

Performs special graphics operations

PC graphics APIs

OpenGL

1992

Cross-platform

No longer under development

Only one thread issuing graphics commands

OpenGL ES (embedded systems)

2003

Cross-platform

Up to date with OpenGL

For mobile systems

WebGL

2011

Cross-platform

For doing OpenGL in browsers

Based on OpenGL ES

DirectX

1995

Microsoft’s graphics api

Vulkan

2016

Cross-platform

Successor to OpenGL

Not backwards compatible with OpenGL

Lower overhead, more fine-grained control than OpenGL

Same Api for PC and mobile

No longer a global state, per object state instead

Metal

Apple’s graphics API

Only for apple devices

WebGPU

Shading language based in Rust instead of a C language

Unit 2

Line drawing algorithms

Digital differential analyzer

How many points (units) a line takes

If ∆X > ∆Y then number = |X∆| + 1, else number = |∆Y| + 1

Bresenham

Difference of distances using a decision parameter Pk

Slope between 1 and 0

Pixels are at integer coordinates, their center is at half-integer coordinates

Pk is the current decision parameter

Pk+1 is the next decision parameter

D1 is the distance from the y-intercept at Pk+1 at yk: y – yx or m(Xk + 1) + b – yk

D2 is the distance from the y-intercept at Px+1 **+ 1**: (yk + 1) – y or yk + 1 – m(xk + 1) – b

The next pixel is xk+1 = xk + 1

Yk+1 = d1 – d2 ≥ 0? Yk + 1 else yk

…

Pk = 2∆y(xk)- 2∆x(yk) + c

c = 2∆ y+∆x(2b -1)

How we use it

If pk ≥ 0 then yk+1 = yk (+1)

If pk < 0 then yk+1 = yk

Midpoint

Two-step

Symmetric two-step

Line drawing project

Write bresenham line drawing algorithm

Make an animation with it

Function render() {

graphics.drawline(x, y, x2, y2, ‘rgb(0, 0, 255)’)

Bresenham cont

P0 = 2∆y-∆x

If pk ≥ 0 then yk+1 = yk + 1

Pk+1 = pk + 2∆y – 2∆x

If pk < 0 then yk+1 = yk

Function animationLoop(time) {

update(); //update data

render(); //sets up but doesn’t draw

requestAnimationFrame(animationLoop); //tell the browser to call the function again when it gets a chance

}

function render() {

graphics.clear();

graphics.drawLine(ptCenter.x, ptCenter

//sends commands to the browser command buffer

}

requestAnimationFrame(animationLoop);

Rasterization

Taking an image described in a vector format and converting it to a raster image

Get outline of triangle and draw horizontal lines to fill it between pixels that touch the edge of the triangle

Curves

Better approximate of natural shapes

Better describe the motion of objects

Curve design criteria

Local control of shape

Smoothness and continuity

Ability to evaluate derivatives (or directly provide slopes)

Stability (Runge phenomenon)

Ease of rendering

Parametric line form

p(u) = (1 – u)p0 +up1

smaller u = more precision

x(u) = axu­3+b …

matrix form

x(u) = [u3 u2 u 1] [ax bx cx dx] or U\*Cx

y(u) = [ “ same with y’s]

piecewise

different geometry between p0 – p1 and p1 – p2

needs points between p0 – pn

slope = dy du / dx du

runge phenomenon

higher order polynomials have more curves between two points

curve techniques

hermite curve (spline)

interpolating (curve goes through the control points), piece-wise, cubic polynomial, where the tangent is specified at each control point

find abcd for x and y

x(0) = p(0)x

x(1) = P(1)x

P(u) = MH \* Mg

Parametric form : x(u) = U\*MH\*Mgx | y(u) = U\*MH\*Mgy

cardinal spline

interpolating, piece-wise, cubic polynomial, where the slope is computed from points adjacent to the curve control points

-also known as catmull-rom spline/curve when t=0

-very similar to the hermite splines, instead of specifying slopes, computed from adjacent points

Compute Pk, Pk+1, Pk-1, Pk+2

Goes through k and k+1 but not necessarily k-1 and k+2

General form: P(u) = U\*Mspline \*Mgeometry

M for matrices

s = (1 – t)/ 2

Bezier curve

Interpolating spline (through start and end points but not intermediate)

Developed by pierre Bezier for use in designing Renault automobile bodies

Because of their properties, they are used to define the curves for fonts and used for font rendering

Can have any number of control points, but usually kept to 3 or 4

Bernstein polynomials

A Bezier curve of 4 points has degree 3

C(n, k) = n! / k!(n-k)!, only computed once

4/16

Assignment 2

Hermite, cardinal, Bezier \* 2

Start with hermite

Every equation goes once with respect to y and once for respect to x

Type

Control points: tangents

How many segments it’s broken into

Bezier

Blending function: A picture containing text, font, white, graphics

Description automatically generated

(once for x and once for y)

Pk = control point for x

U based on number of segments

N = degree of curve ( 4 points = degree 3)

K = which degree we’re working on

Cardinal spline

Doesn’t use tangents

Pk -1 not on the curve pk + 2 not on the curve

Multiply matrix once for y and once for x

Tension, how closely the curve is to the points

Hermite

Uses tangents

Line drawing review

Combine p0 into loop

Pass in x increment (-1 or 1)

Closure and memo-ization

Let factorial = function () {

Let f = [];

Return function inner(n) {

}

}();

//outer function executes immediately, inner function assigned to factorial

//previously computed values stored in f

Memo-ize C(n, k)

Let BlendC = function () {

Let memo = [];

Function compute(n, k){

Return factorial(n) / (factorial(k) \* factorial( n – k));

}

Return function inner(n, k) {

if (n > memo.length -1) {

memo[n] = [];

memo[n][k] = compute(n, k);

}else if (n < memo.length && k > memo[n].length – 1) {

Memo[n][k] = compute(n, k);

}

return memo [n][k]

}

//n will always be 3 for a cubic

Precompute with for loop up to 3

For (let k = 0; k < 3; k++) {

BlendC(3, k);

}

Let respectToU(u) {

Let memo = []

Return function inner(u) {

Let u1000 = Math.trunc(u \* 1000);

If(memo[u1000] == undefined) {

Memo[u1000] = compute(u);

}

Return memo[u1000];

}

//Use to precompute (2u^3- 3u^2 + 1) plus all ‘u’ sections

//precompute to memorize before going to render?

//code without memo-ization first then add later

//bresenham equation in quiz 2

4/18

Tension can be part of control

Affine Transformations

Translation

Moving one point by a vector giving us a new point

V(x, y) = P’ (x, y) – P(x, y)

dx = x’ – x

dy = y’ - y

Scaling

Change size

Scale each point P by a scaling factor S(x, y) resulting in P’

Each point is scaled in relation to the object’s center

S(x) > 1 moves away from center

S(x) < 1 moves towards the center

Rotation

Most complex

Rotate around a pivot point

Rotate by angle theta between two vectors originating from pivot

X = r cos(fi)

Relative to axis origin not object origin

Global coordinates

Translate object to be on axis origin, rotate, then translate back out

Local coordinates

Rotate around a local origin, then translate by world coordinate at render

Homogenous coordinates

Translation P’ = I \* P + T

H = 1, then cartisian = homogenous

Lets translation be a matrix multiplication instead of addition

Multiply rotation, translation, and scale steps in reverse order of what you want to happen to get single matrix that does all steps

P = Mc \* P (Mc translates in, rotates, then translates back out, times the points P)

Rotate about a line

P’ = T2 \* R0 \* Rx \* R-0 \* T1 \* P

0 is theta, put line intercept at origin, rotate line into x axis, rotate then undo rotate and translate.

Vector space

Euclidian space

Extends vector space by adding size and distance (magnitude of a vector)

Affine space

Extends vector space by adding a point

Preserves points, straight lines and planes

Parallel lines remain parallel

Angle and distances aren’t necessarily preserved

function drawLineQ3(x1, y1, x2, y2, color) {

var deltaY = Math.abs(y1 - y2);

var deltaX = Math.abs(x1 - x2);

var pk = (2 \* deltaX) - deltaY;

var curY = y1;

var curX = x1;

if(color === 'red'){

api.octant1 = 3

}else{

api.octant2 = 3;

}

for(let i =0; i<deltaY; i++){

//d1 - d2 = positive go up, else go down

api.drawPixel(curY, curX, color);

if(pk >= 0){

curX++;

curY++;

//pk+1

pk = pk + 2 \* deltaX - 2 \* deltaY;

}else{

curX++;

//pk+1

pk = pk + 2 \* deltaX;

}

}

}

4/23

**Computation is fast but memory is slow**

Cpu

Looks for data in cache

Cache

Not in cache – stall

Main memory

Array

cpu grabs cache line, some data around the main data if it’s in the line

Javascript

Can accidentally grab objects from array one at a time with cache stall for each object

Much slower than bringing in the whole array and then indexing

Assignment 3

Primitive – anything with a bunch of points (2 or more vertices)

Connect – if line goes from last point back to first

Local – center x, y is 0, 0

Center and verts can be local or world coordinates

Only translate primitive’s center and have other points update

Rotate in world – have to translate in and back out

Hermite not required just cardinal and Bezier

Self define what center means

Left handed coordinates system

X cross product into Y gives positive Z

Positive Z goes ‘away’ from you

Right handed coordinates system

X cross product into Y to get positive Z (goes towards you)

OpenGL – righthanded with some lefthanded

Direct x – lefthanded

Translation in 3D

T = 100t1

010t2

001t3

0001

Rotation in 3D

Rotation in x, y plane, x, z plane, y, z plane

About z-axis: x-y plane

About x-axis: y-z plane

About y axis: z-x plane

x-y plane (righthanded)

X’ = xcos(0) – ysin(0) (0 = theta)

Y’ = xsin(0) + ycos(0)

Z’ = z

y-z plane

x’ = x

y’ = ycos0 – xsin 0

z’ =

Rotation about a line 3D

Line endpoints pa – pb

1. Translate pa to origin t1
2. Rotate line into a coordinate axis (2 rotations), Ra Rb
3. Rotate by the specified ø about the coordinate axis
4. Rotate back by negative rb ra
5. Translate back by -t1

P’ = -t1\*r-a\*r-b\*r0\*rb\*ra\*t1\*P

Gimbal lock

Point gets stuck in two access planes and can’t mathematically get out

Need 4-dimensional complex number to do rotation in 3D but 2-dim complex to do 2D rotation

Because 3-dimensional complex operations are undefined

-quaternion math

Arbitrary 3D rotation

Can be done on the GPU