COMS 30115

Rasterisation

Carl Henrik Ek - carlhenrik.ek@bristol.ac.uk

February 16th, 2018

http://www.carlhenrik.com

Last time

- Shadows
- How to think about optimisations in raytracing?
    - Datastructures
    - Instruction level
    - Fidelity level
- Summarisation of Raytracing

Today

- Start with Rasterisation
- Image space rendering
- Line drawings

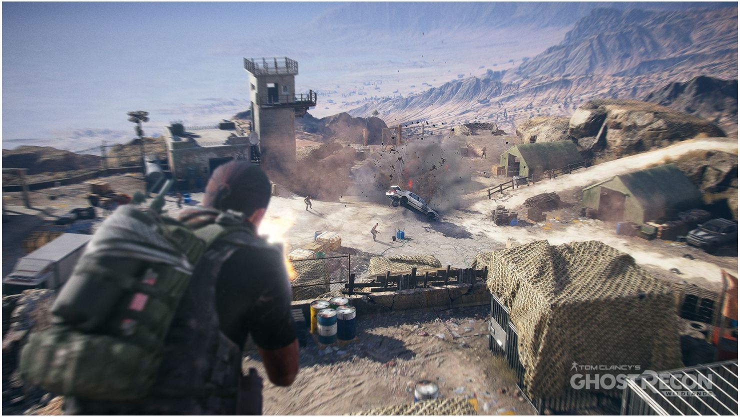
The Book

sadly the books is rather empty on the material in this lecture

- Breshenham URL
- Paper on line drawing algorithms URL

Rasterisation

Real-time Graphics[1]



[1] Ghost Recon Wildlands

Real-time Graphics[1]

- Remember the generative model?
  $I = f(x)$
  - how much data is actually I?
- 50 HD images/second

$$50 \times 1920 \times 1080 \times 4 \approx 400mb/s$$

- Interactive ⇒ Latency important
- Minimum and consistent performance important

Real-time Graphics

- Efficient implementations
  - don’t think how does physics work, think of the effect physics has
  - emulate the effect in a simpler way
- Adapt to hardware
  - write code that respects the hardware

Programming

Abstraction is the enemy of efficency
– Carl Henrik

C/C++ [2]

- *"the most advanced assembler on the market"*

[2] Mikael Kalms, (former DICE)

C/C++ [2]

- *"the most advanced assembler on the market"*
- Available on nearly all machines

[2] Mikael Kalms, (former DICE)

C/C++ [2]

- *"the most advanced assembler on the market"*
- Available on nearly all machines
- Can access native APIs on nearly all OSes

[2] Mikael Kalms, (former DICE)

C/C++ [2]

- *"the most advanced assembler on the market"*
- Available on nearly all machines
- Can access native APIs on nearly all OSes
- Control over memory management

[2] Mikael Kalms, (former DICE)

C/C++ [2]

- *"the most advanced assembler on the market"*
- Available on nearly all machines
- Can access native APIs on nearly all OSes
- Control over memory management
- Control over memory layout

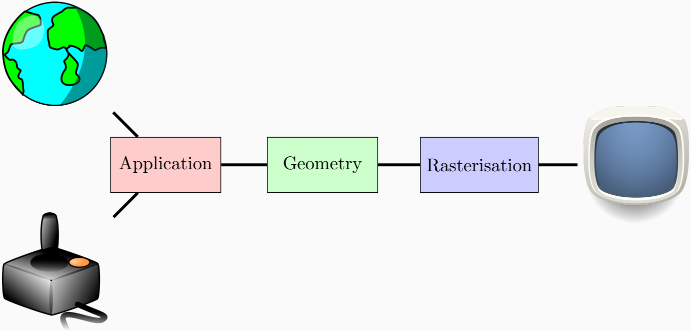
[2] Mikael Kalms, (former DICE)

C/C++ [2]

- *"the most advanced assembler on the market"*
- Available on nearly all machines
- Can access native APIs on nearly all OSes
- Control over memory management
- Control over memory layout
- No garbage collection

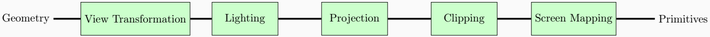
[2] Mikael Kalms, (former DICE)

Rendering Pipeline



- Rasterisation is mainly rastering ;-)

Rendering Pipeline



☒ Transformations (same)

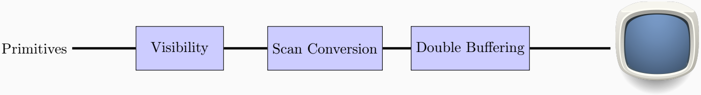
☐ Lighting (Next week)

☐ Projections

☐ Clipping (After explore week)

☐ Screen Mapping (Today:ish)

Rendering Pipeline

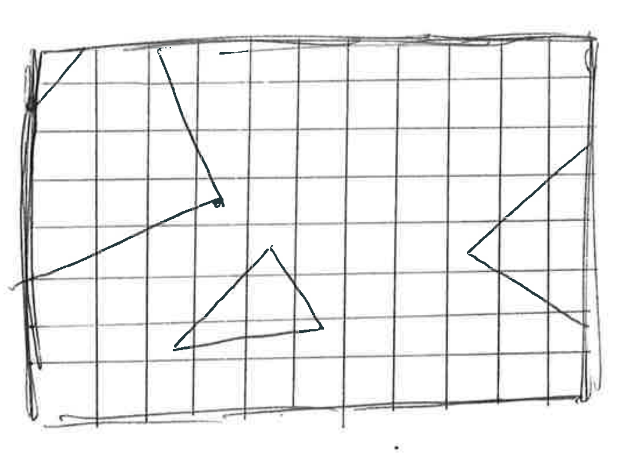


☐ Visibility (Next Week)

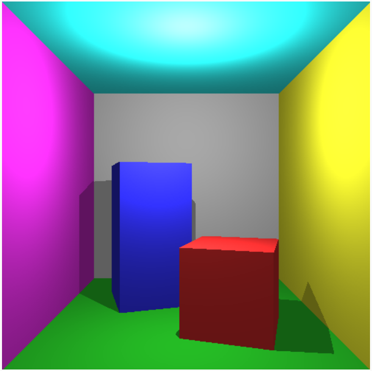
☐ Scan Conversion (Today and Monday)

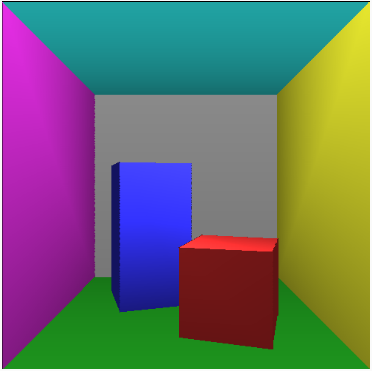
☒ Duble Duffering (Same)

Rendering Pipeline

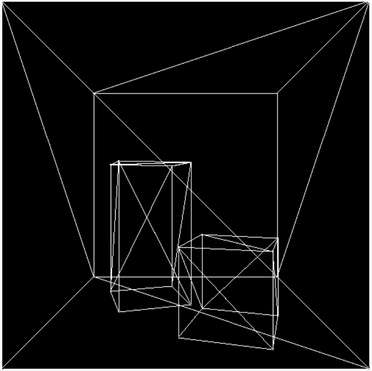


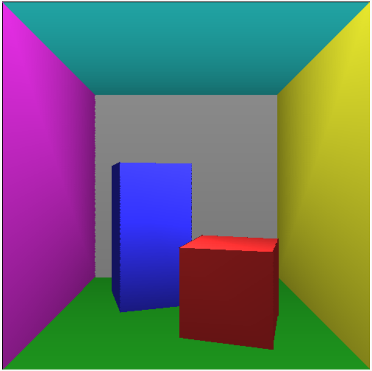
Part I vs Part II





General Concept





General Concepts

- Raytracer simple and slow
  - Calculate shading in the "world"
  - *What part of the world matches each pixel?*
  - Calculations in world

General Concepts

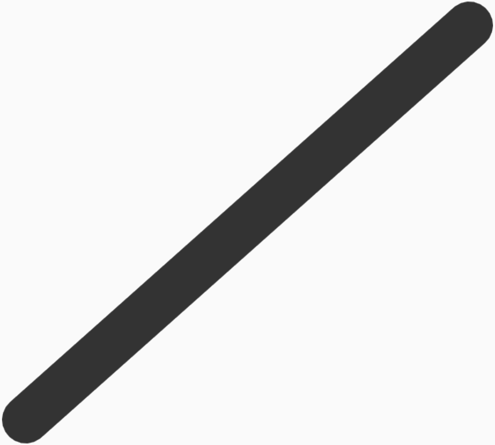
- Raytracer simple and slow
  - Calculate shading in the "world"
  - *What part of the world matches each pixel?*
  - Calculations in world
- Rasteriser messy but fast
  - Calculate shading in the image
  - *What pixel does this part of the world match to?*
  - Calculations in screen space

Structure of Part II

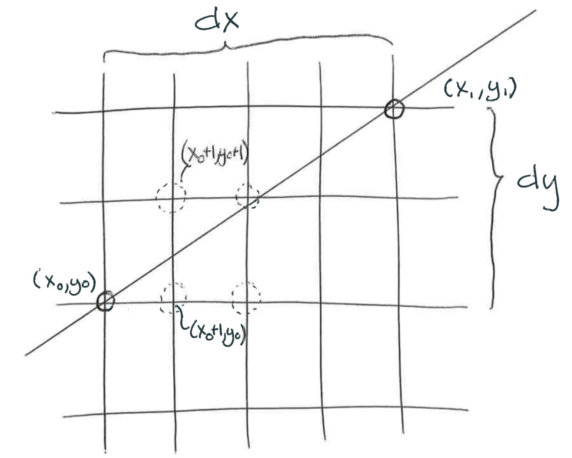
- How to draw in screen space
    - primitives (lines, triangles)
    - how to draw discrete data
- How to do sparse computations
    - interpolation
- How to solve visiblity problem
- Shading in image space (vertex shading)
- Mappings (Texture etc.)

Line Drawing

Line



Digital Differiential Analyser (DDA)



$$y = k \cdot x + m$$

Digital Differiential Analyser (DDA)

```
putpixel(x,y);
dx = x1-x0;
dy = y1-y0;
if(dx>dy){steps = abs(dx);}
else{steps = abs(dy);}
xp = dx/steps; yp=dy/steps;
for(int i=0;i<steps;i++)
{
  x += xp;
  y += yp;
  putpixel(round(x),round(y));
}
```

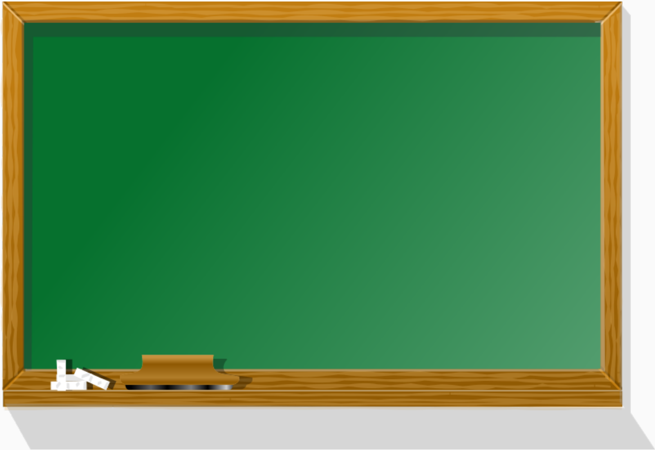
Breshenham's Line Drawing

- DDA algorighm is slow
    - rounding of floats
    - float additions

Breshenham’s Line Drawing

- DDA algorighm is slow
  - rounding of floats
  - float additions
- Optimise by reducing to more cases that are less general

Breshenham's Derivation

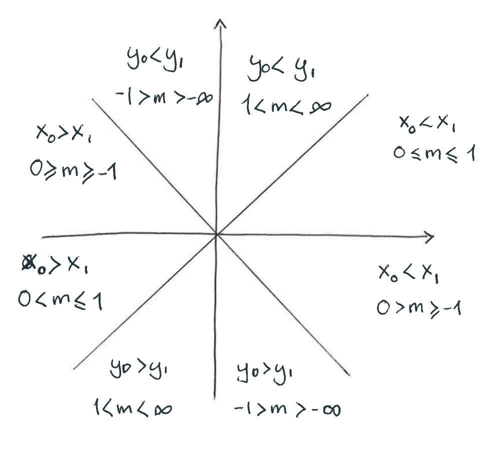


Breshenham's Line Drawing

```
putpixel(x0,y0);
dx = x1-x0;dy=y1-y0;
2dx = 2*dx; 2dy = 2*dy;
2dydx = 2dy - 2dx;
d = 2dy-dx;
```

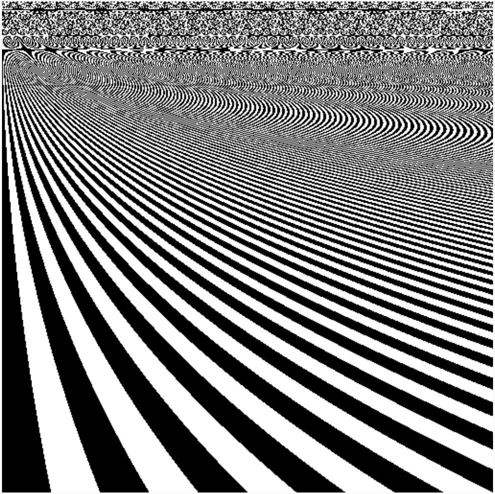
```
for(int i=0;i<dx;i++){
  if(d<0){
    x += 1;
    d += 2dy;
  }
  else{
    x+=1;
    y+=1;
    d += 2dydx;
  }
  putpixel(x,y);
 }
```

Breshenham's Line Drawing

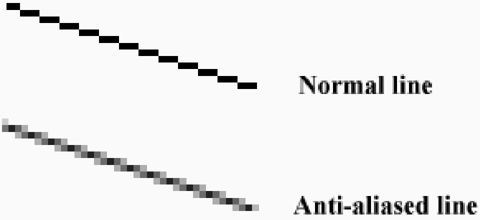


- Completely integer
- 8 (4) cases for lines (really 4+4)

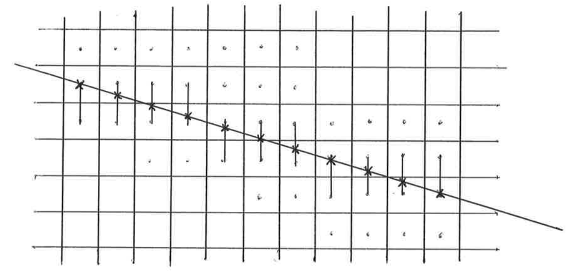
Aliasing



Aliasing



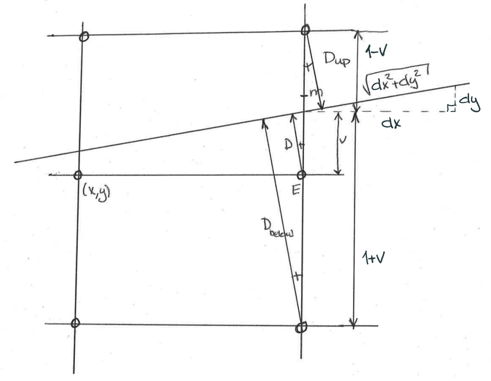
WU Lines[3]



- Compute distance between line and pixel
- Set pixel above and below
- Intensity sum to 1

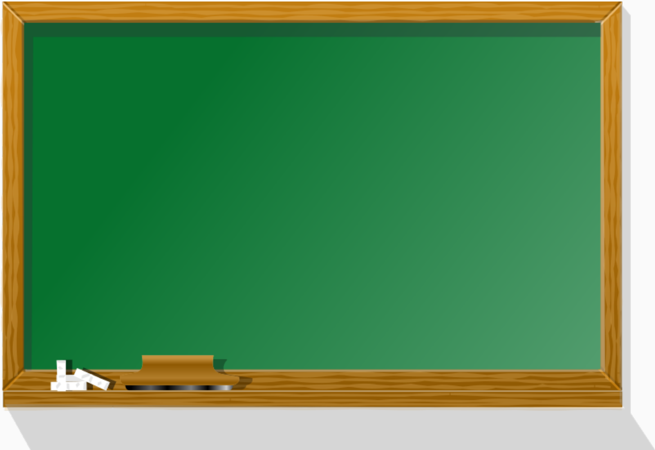
[3] https://en.wikipedia.org/wiki/Xiaolin_Wu%27s_line_algorithm

Gupta-Sproull



- Area that pixel covers important
- Weight pixels with perpendicular distance

Gupta-Sproull: derivation



Gupta-Sproull

```
//compute constants A,B
//1. Run Breshenham and get d
if(d<0) //E pixel
  {
    D = A*(d+dx);
    Dup = B-D;;
    Dbelow = B+D;
    // look-up shading based on D
```

Summary

Summary

- Work in image space
- Make sparse computations and interpolate
- Example of interpolation: Lines
- Graphics is expensive, if we want realtime we have to think
- Optimise by making many simpler cases
    - cheap especially for simple conditions on integers

Next Time

Lecture next primitive

- Triangles
- Perspective correct interpolation

Lab start with Lab 2

EOF

eof