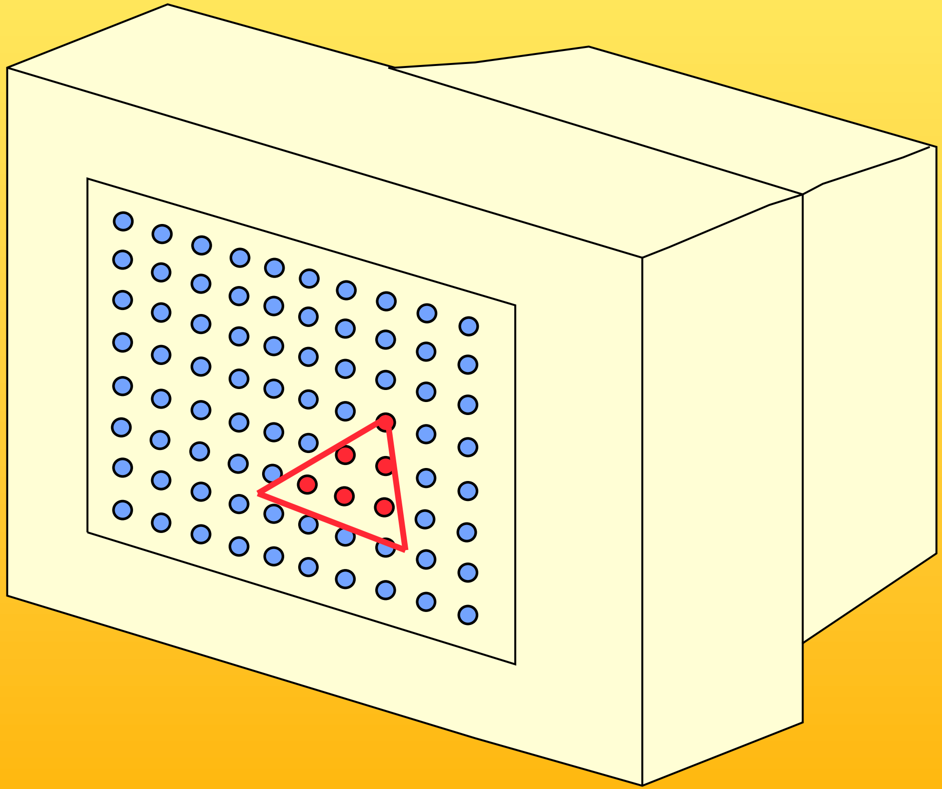
CHALMERS

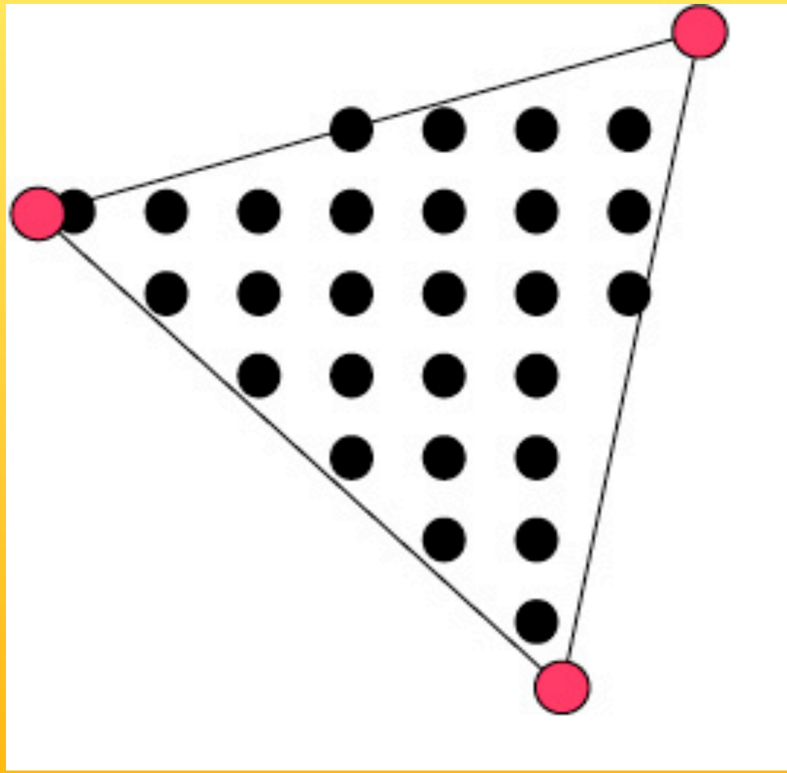
3D Graphics Hardware

Ulf Assarsson

Vovve – 17 fps

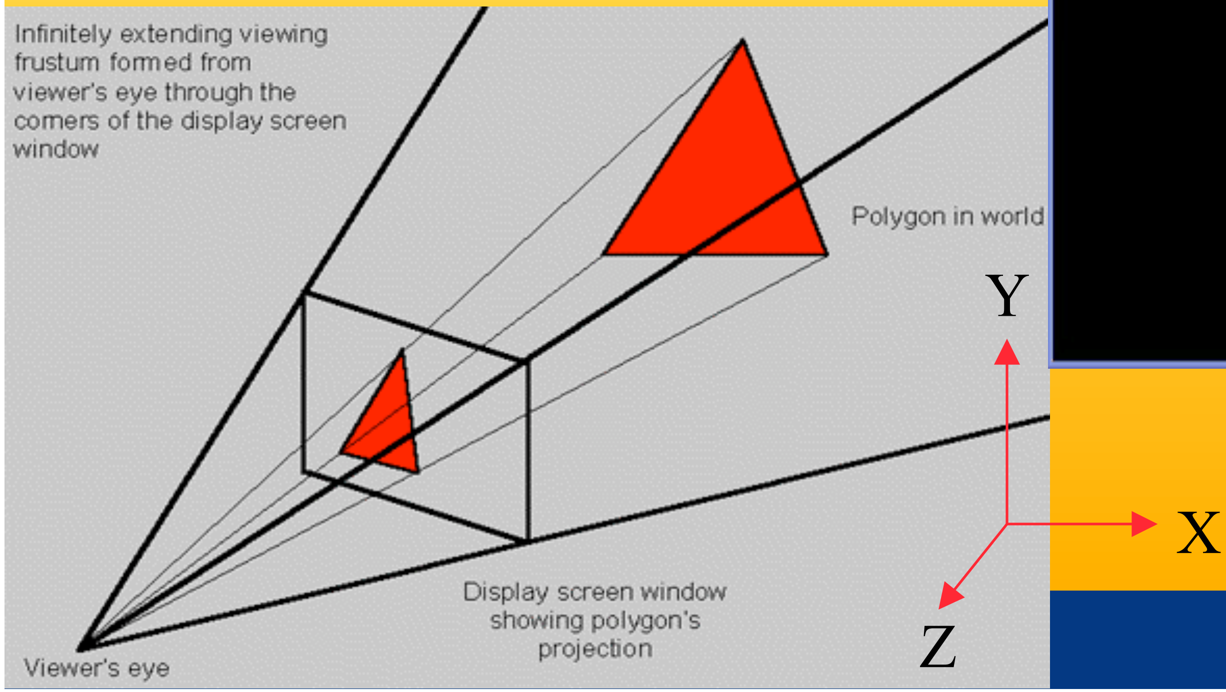
The screen consists of pixels

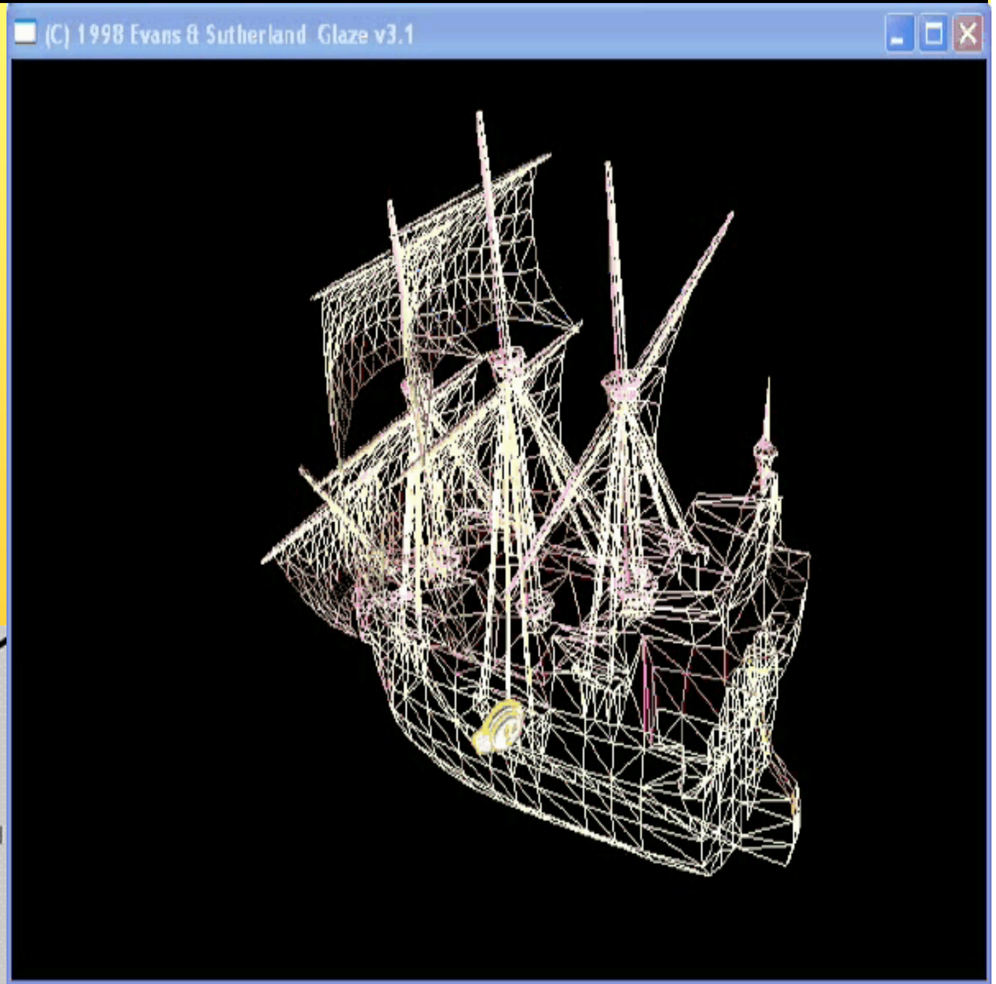




3D-Rendering

- Objects are often made of triangles
- x,y,z- coordinate for each vertex

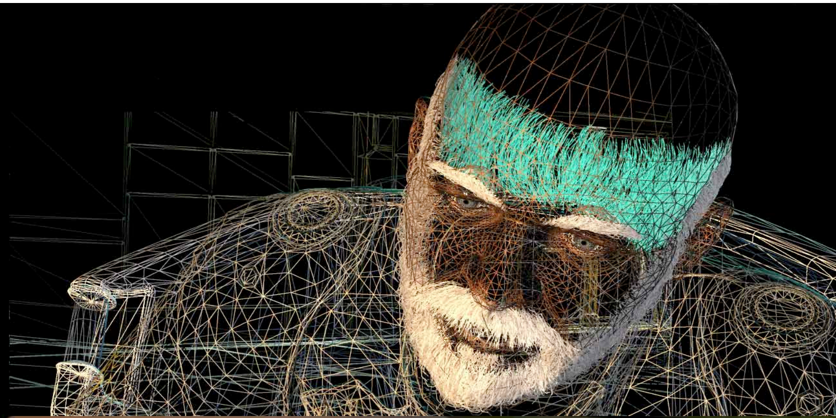




4D Matrix Multiplication

$$\begin{bmatrix} s_x & \bullet & \bullet & t_x \\ \bullet & s_y & \bullet & t_y \\ \bullet & \bullet & s_z & t_z \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ z \\ w \end{bmatrix}$$

Real-Time Rendering

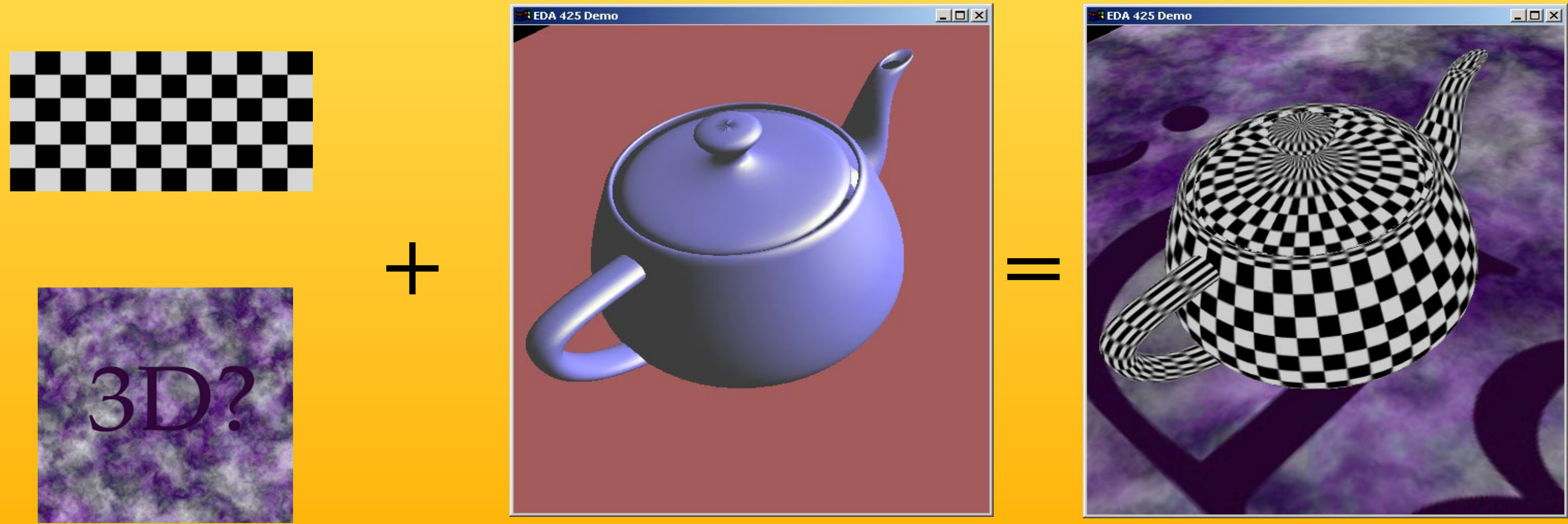


Textures

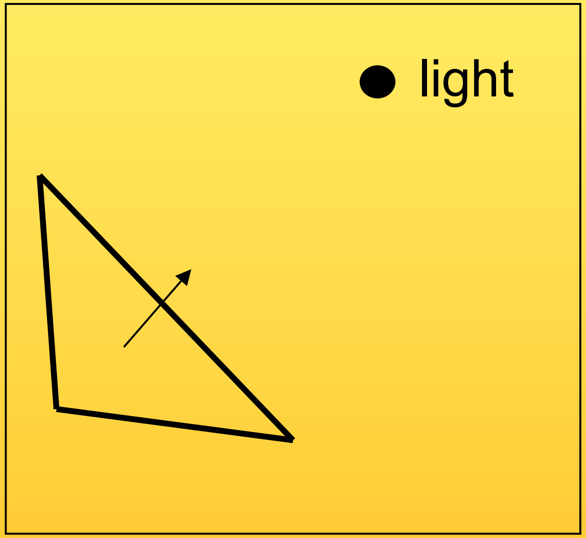
- One application of texturing is to ”glue” images onto geometrical object

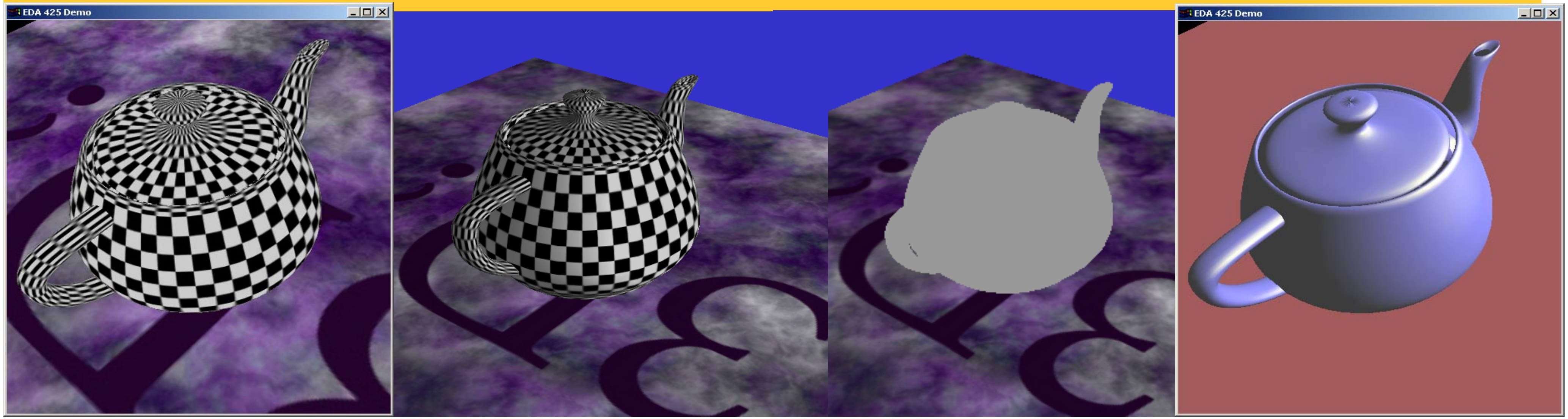
Texturing: Glue images onto geometrical objects

- Purpose: more realism, and this is a cheap way to do it

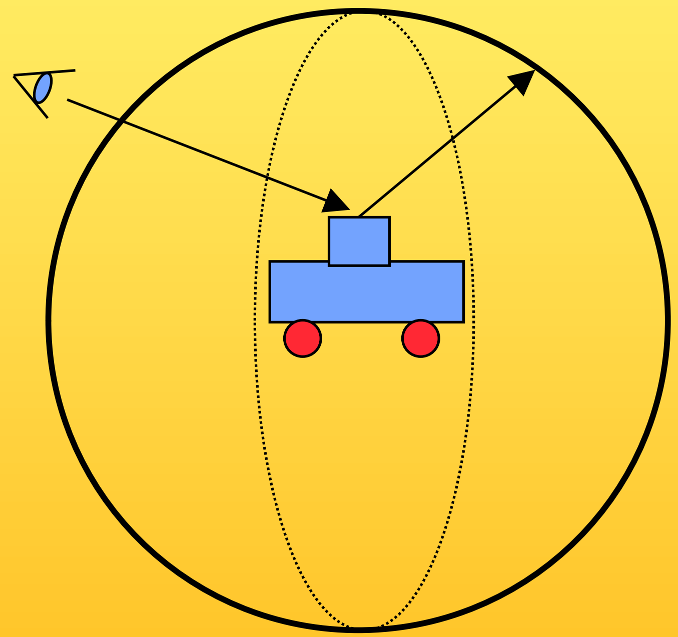


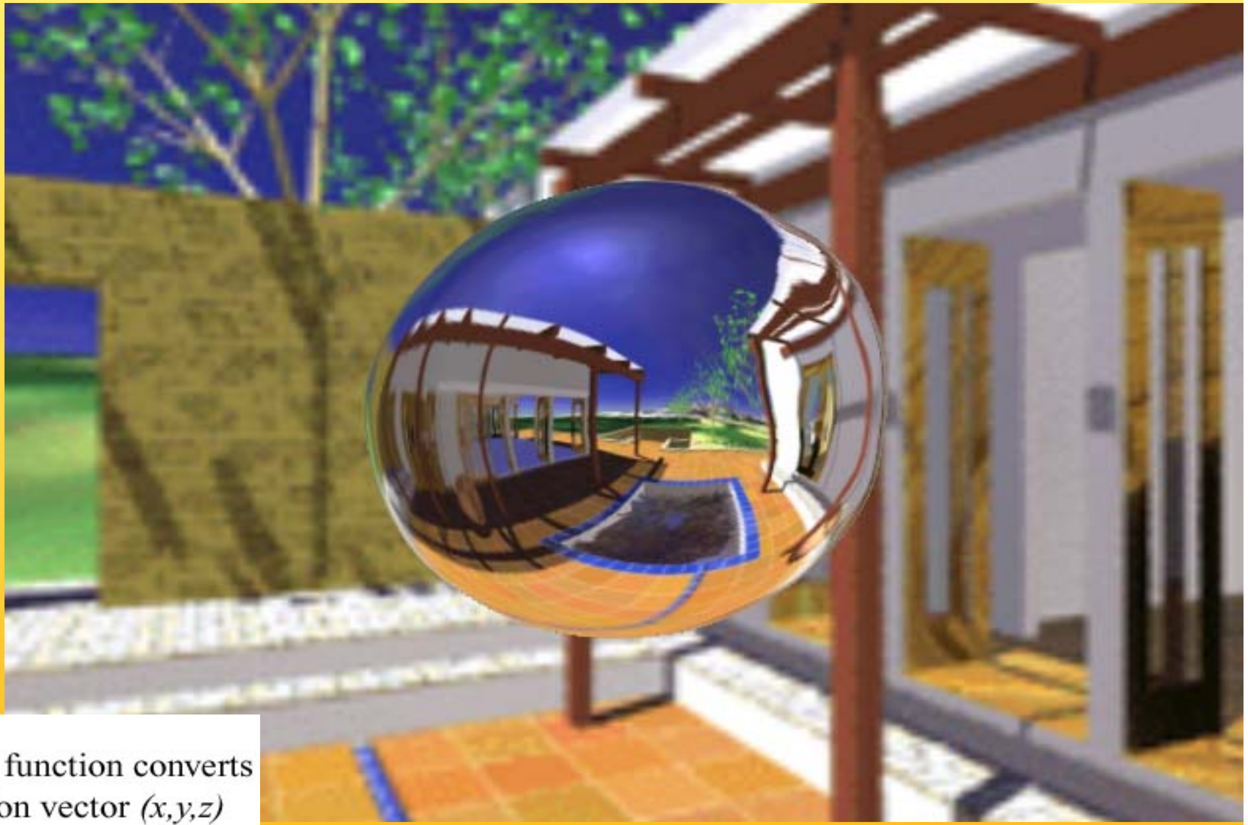
Light computation per triangle

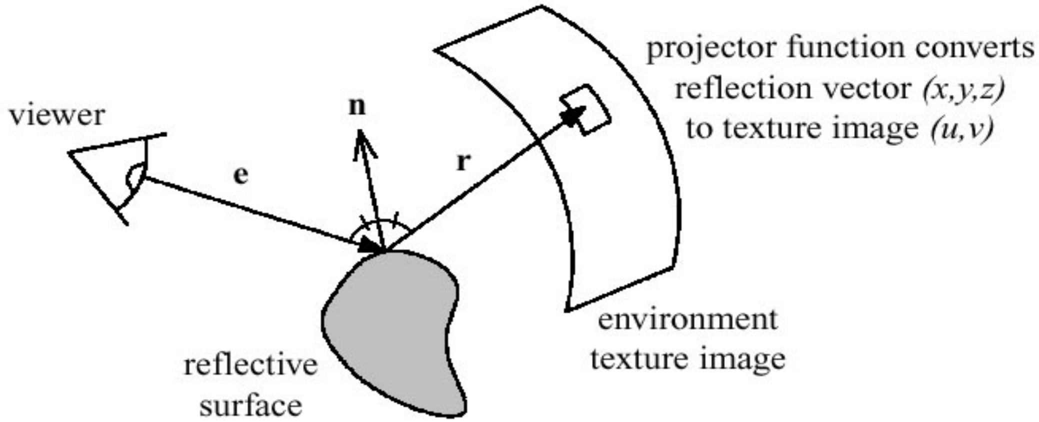




Environment mapping







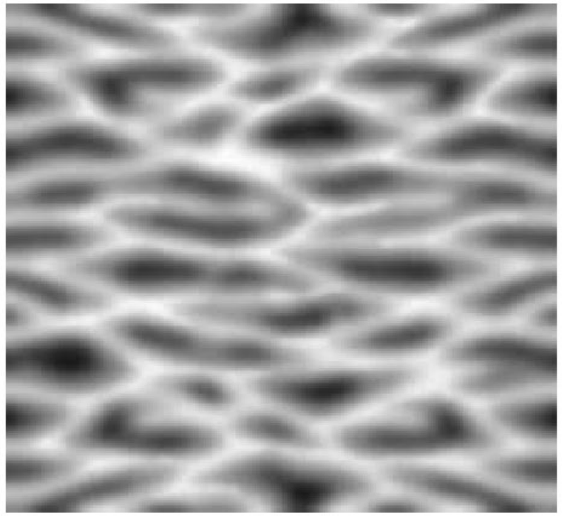
Sphere map

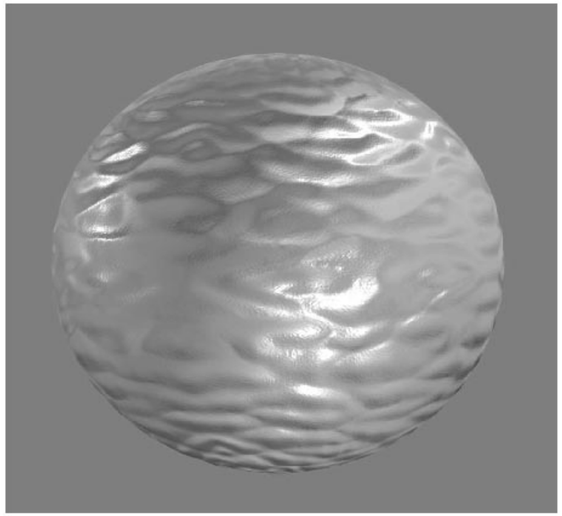
- example

Sphere map (texture)

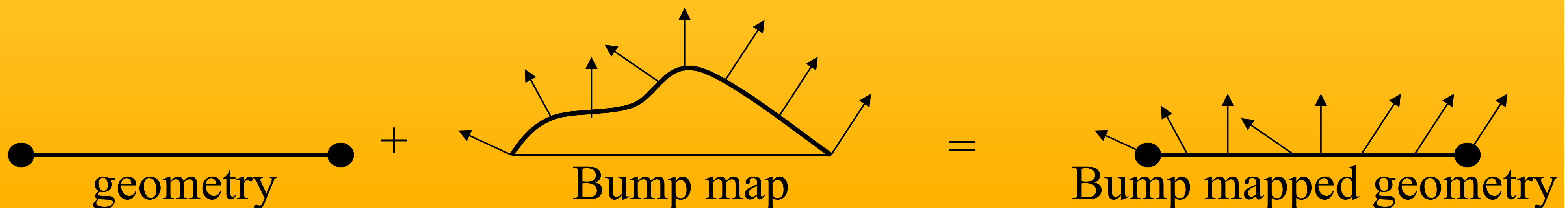
Sphere map applied on torus

Bump mapping



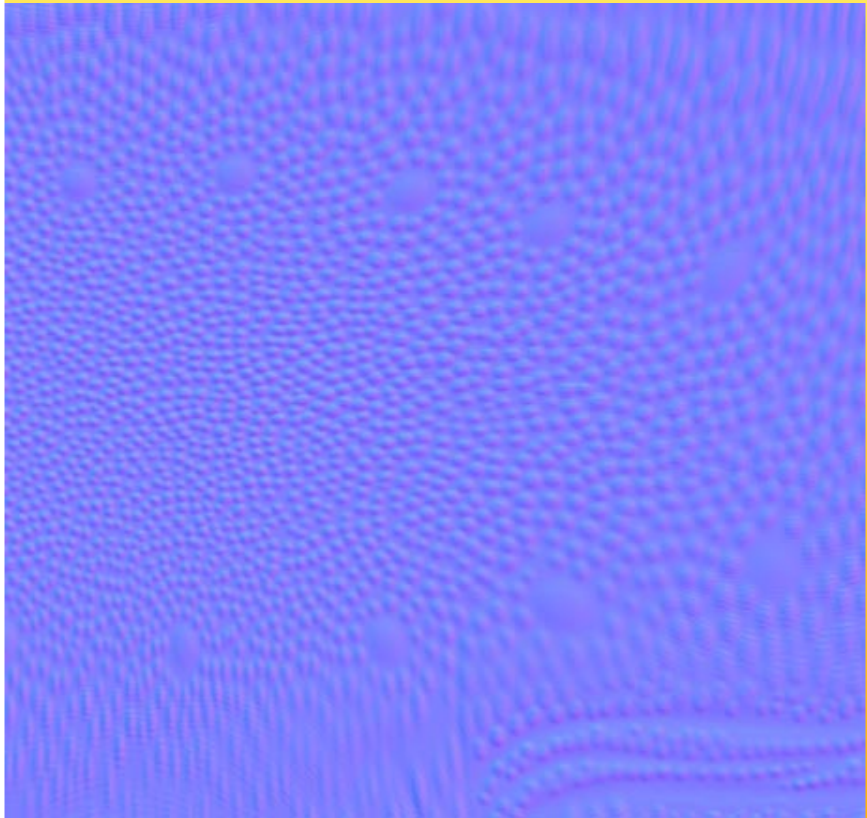


- by Blinn in 1978
- Inexpensive way of simulating wrinkles and bumps on geometry
  - Too expensive to model these geometrically

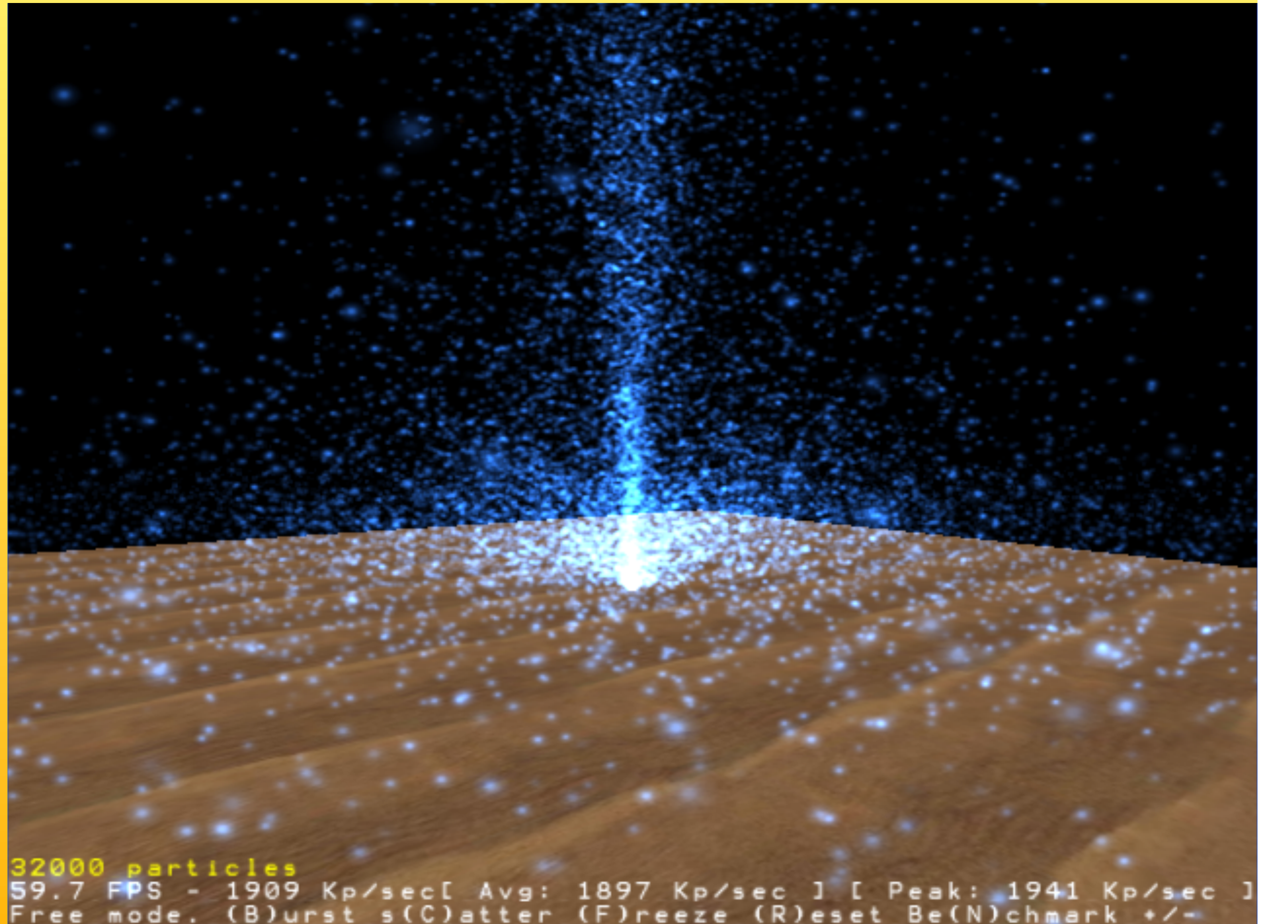


Stores heights: can derive normals

Bump mapping: example



Particle System

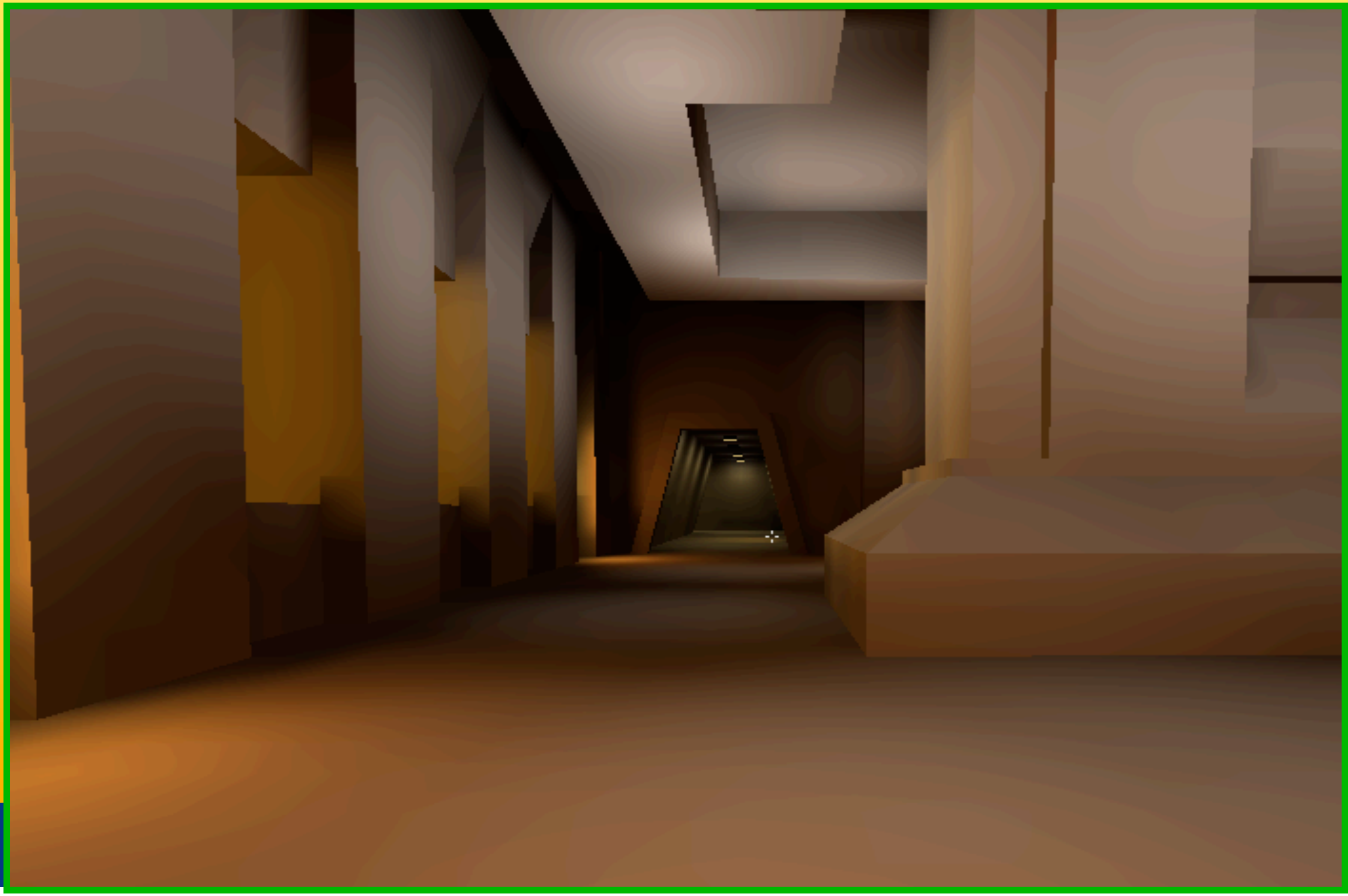


Particles

Shadows

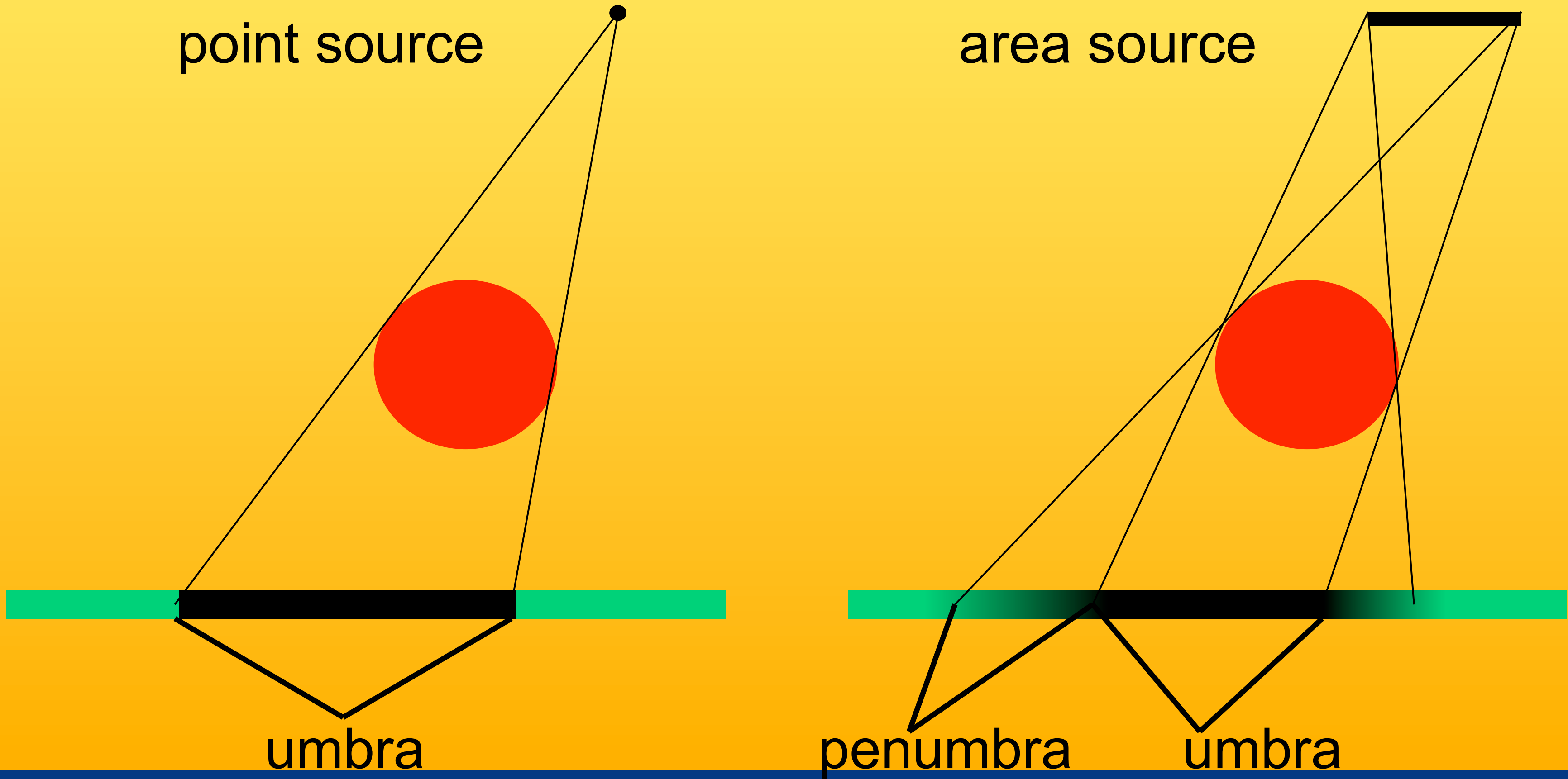
- More realism and atmosphere

Shadows play an important role for realism

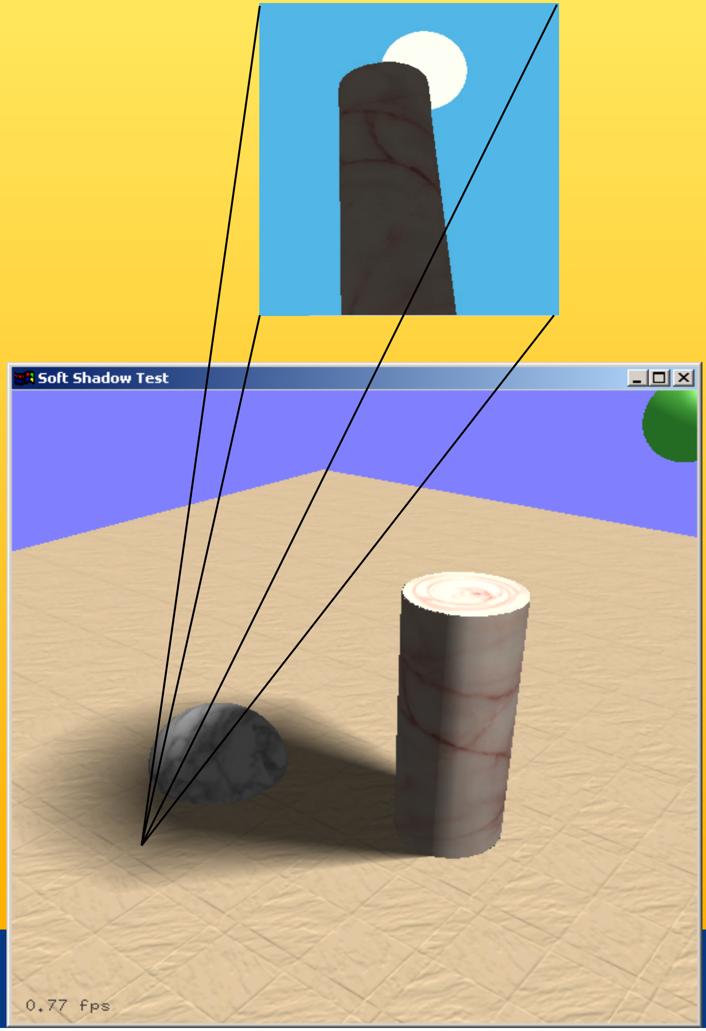


Hard vs. soft shadows

Two different light source types:

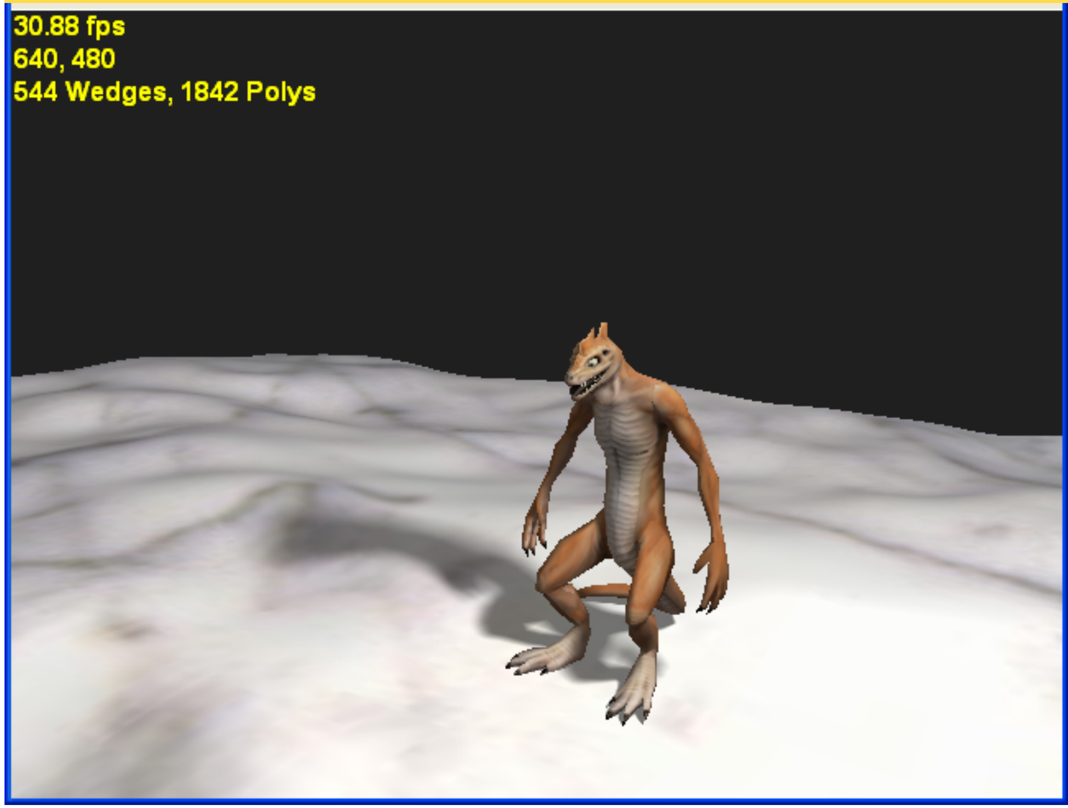


Very brief explanation of the Soft Shadow Volume Algorithm

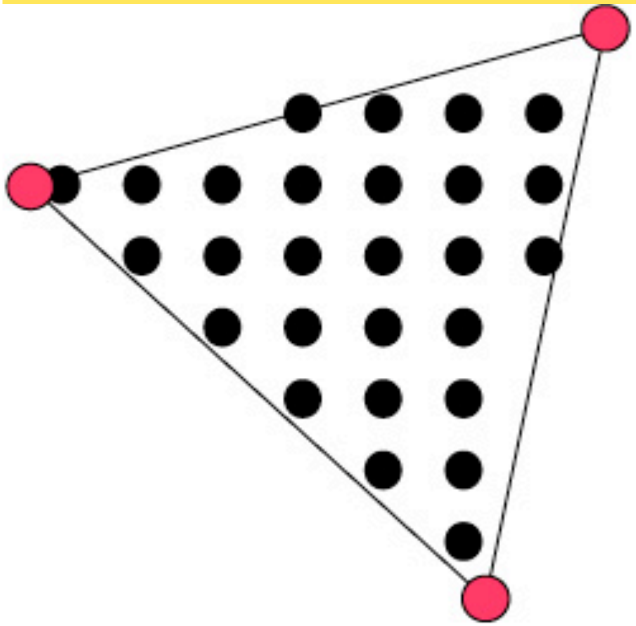


Mjuka skuggor

http://www.ce.chalmers.se/staff/tomasm/soft/



What is vertex and fragment (pixel) shaders?



- Memory: Texture memory (read + write) typically 256 Mb – 1GB
- Program size: unlimited instructions (but smaller is faster)
- Instructions: mul, rcp, mov,dp, rsq, exp, log, cmp, jnz…

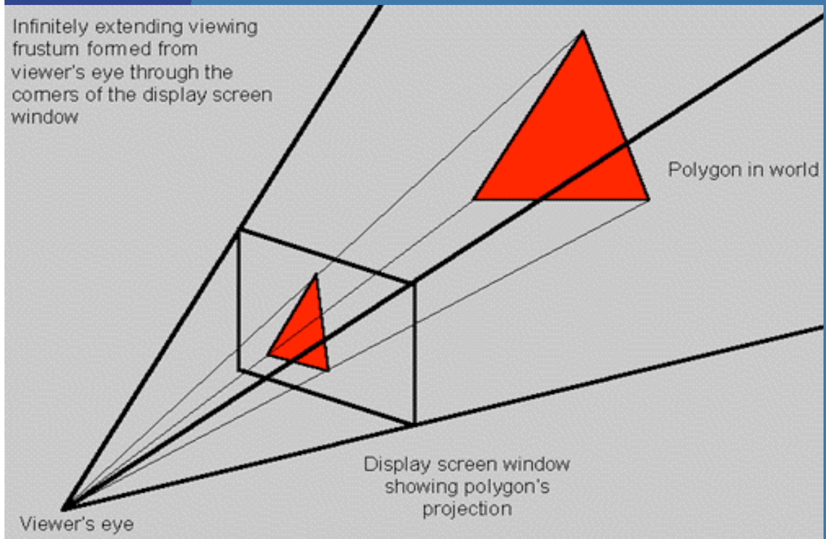
For each vertex, a vertex program (vertex shader) is executed

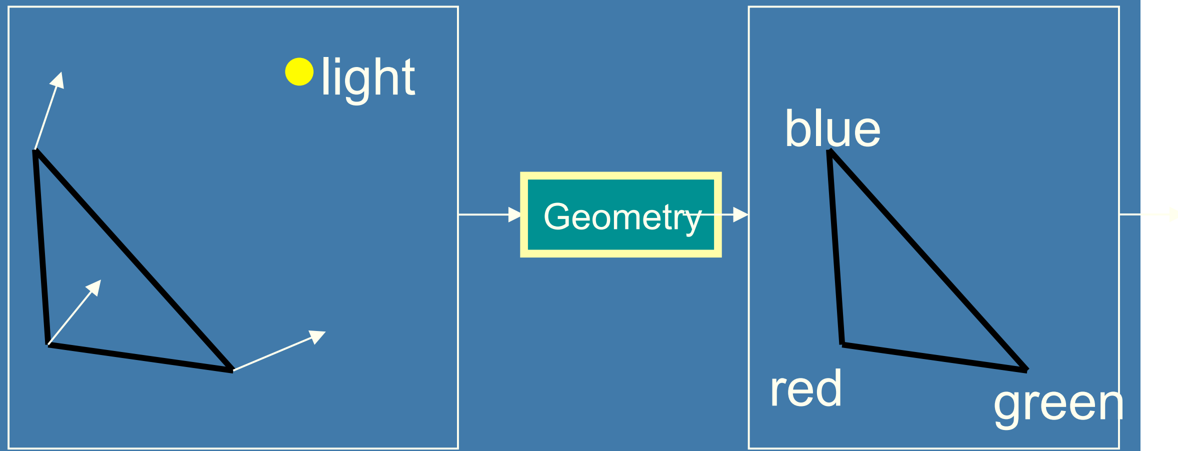
For each fragment (pixel) a fragment program (fragment shader) is executed

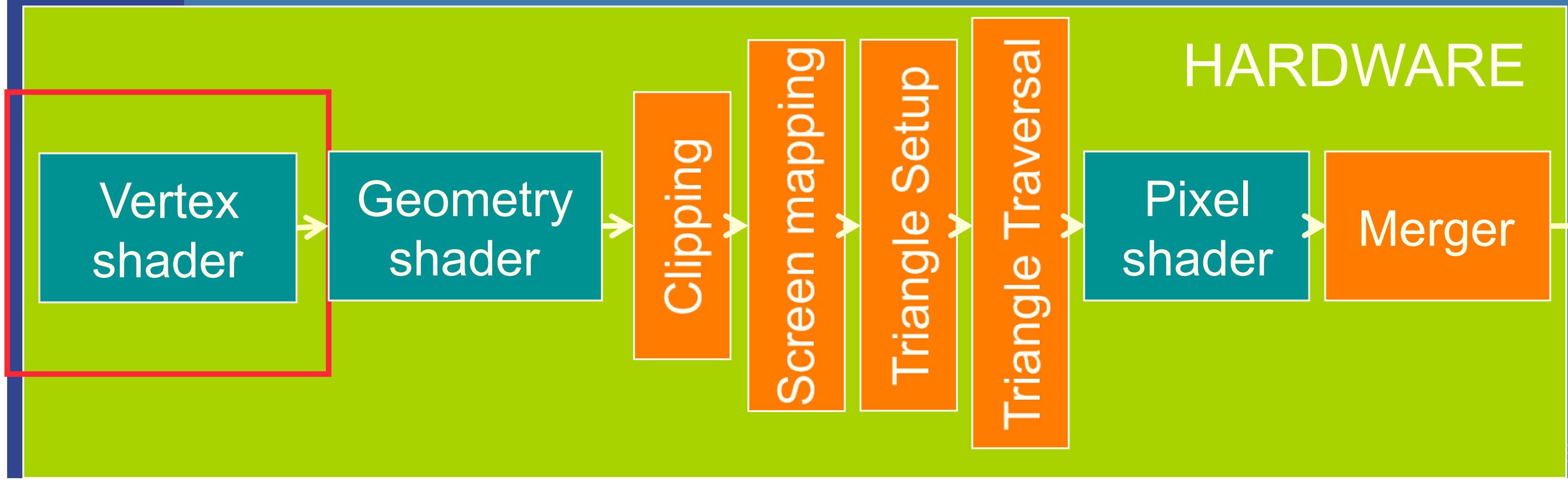
Hardware design

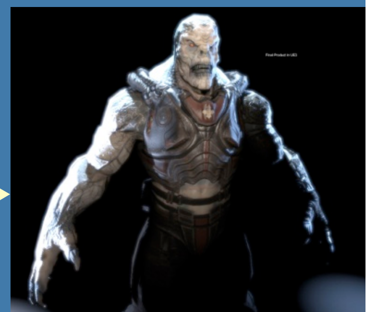
Vertex shader:

- Lighting (colors)
- Screen space positions









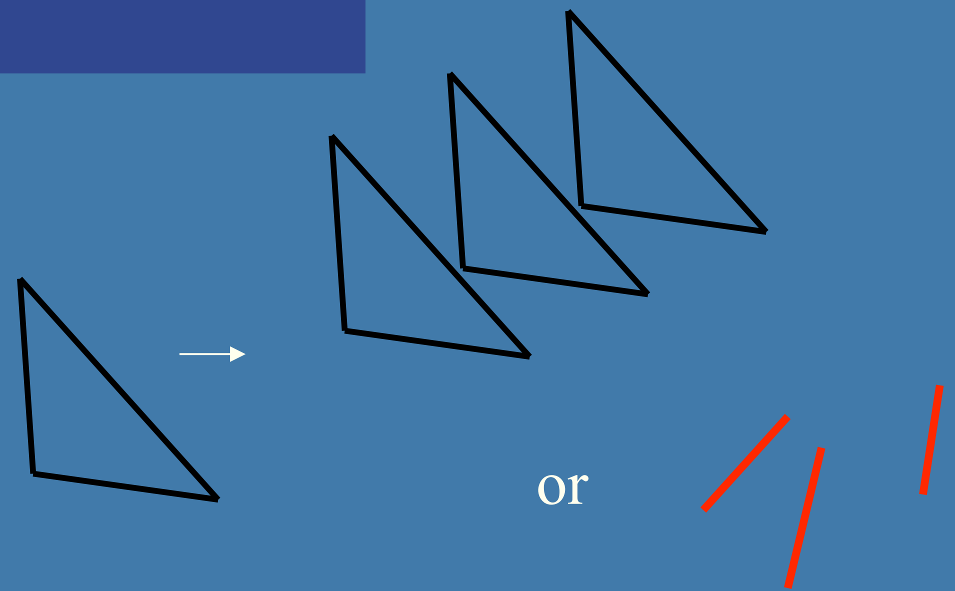
Display

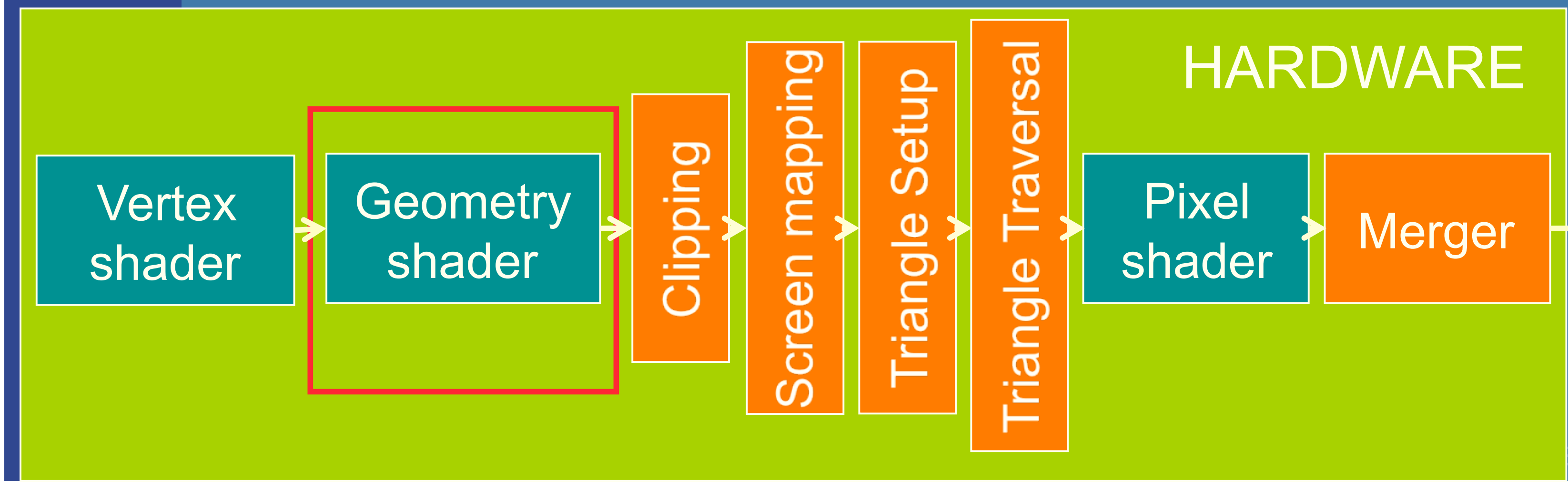
Hardware design

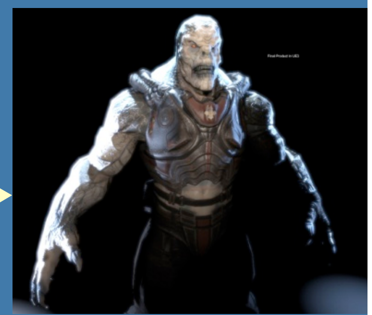
Geometry shader:

•One input primitive

•Many output primitives

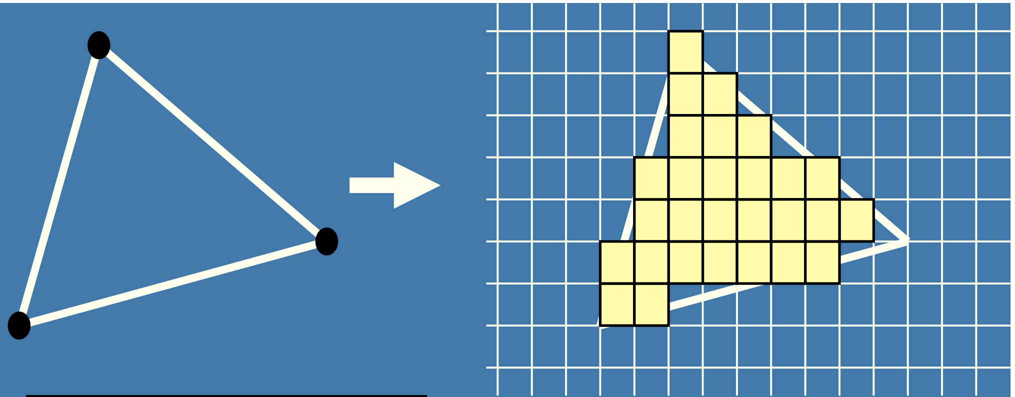


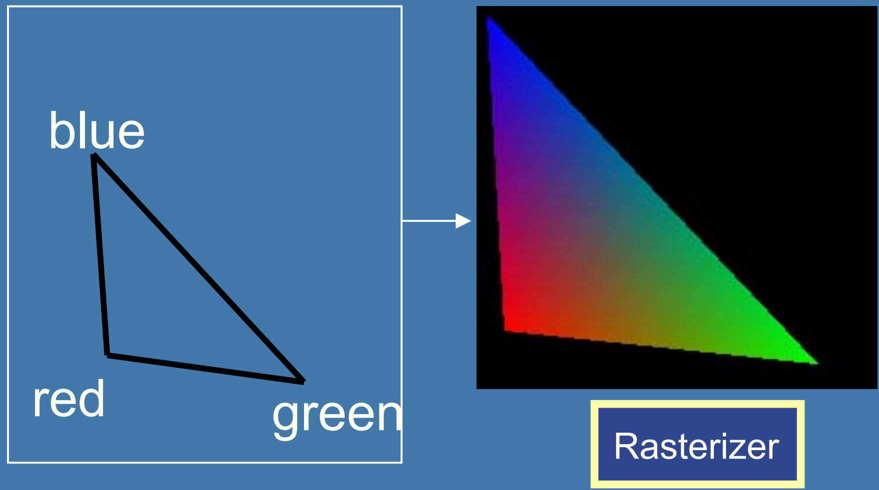




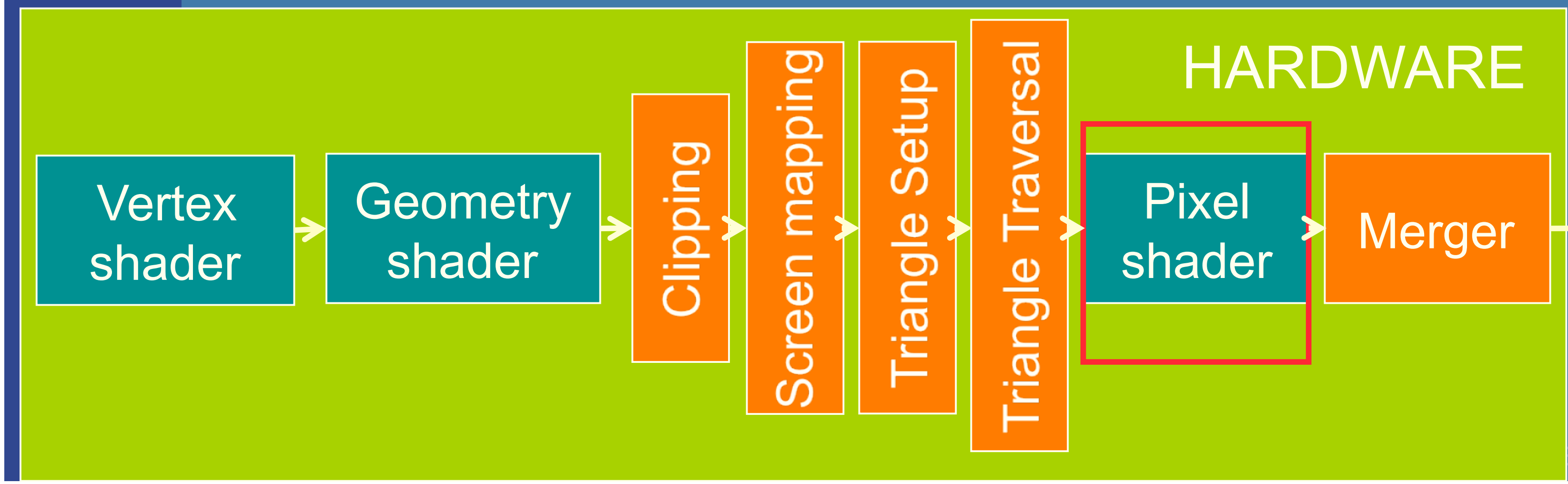
Display

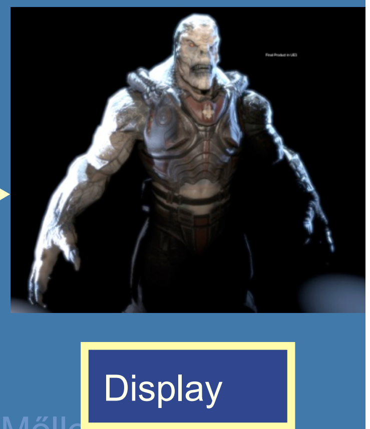
Hardware design



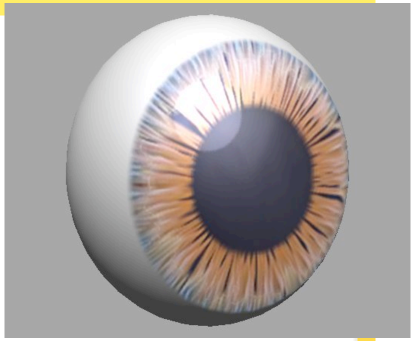


Pixel Shader:
Compute color using:
- Textures
- Interpolated data (e.g. Colors + normals)



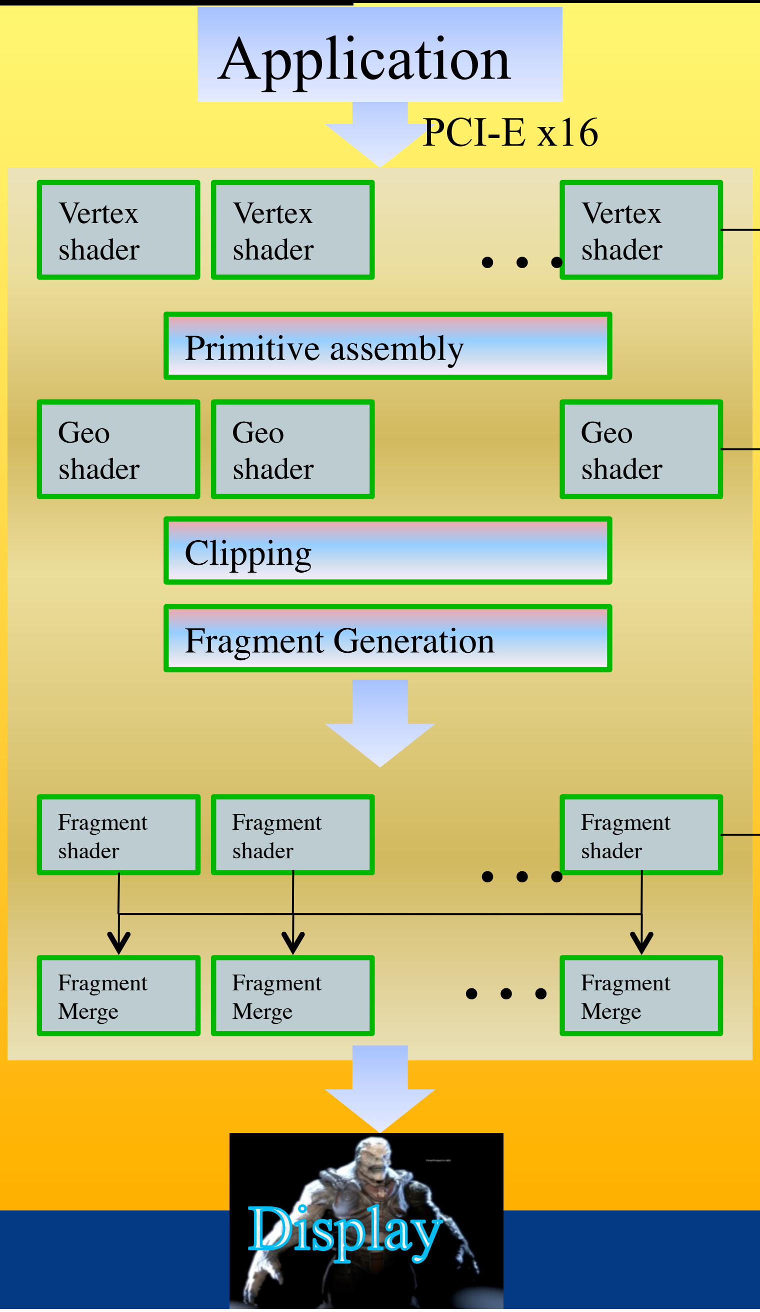


Cg - "C for Graphics" (NVIDIA)



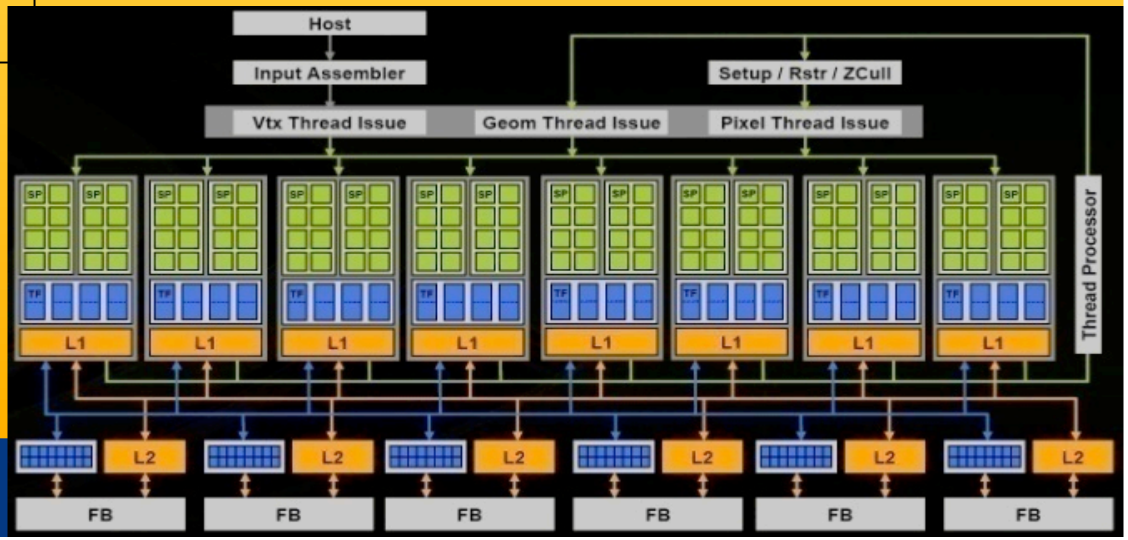
```
if (slice >= 0.0h) {
     half gradedEta = BallData.ETA;
     gradedEta = 1.0h/gradedEta;  // test hack
     half3 faceColor = BgColor; // blown out - go to BG color
     half c1 = dot(-Vn,Nf);
     half cs2 = 1.0h-gradedEta*gradedEta*(1.0h-c1*c1);

     if (cs2 >= 0.0h) {
          half3 refVector = gradedEta*Vn+((gradedEta*c1+sqrt(cs2))*Nf);
          // now let's intersect with the iris plane
          half irisT = intersect_plane(IN.OPosition,refVector,planeEquation);
          half fadeT = irisT * BallData.LENS_DENSITY;
          fadeT = fadeT * fadeT;
          faceColor = DiffPupil.xxx;  // temporary (?)
          if (irisT > 0) {
               half3 irisPoint = IN.OPosition + irisT*refVector;
               half3 irisST = (irisScale*irisPoint) + half3(0.0h,0.5h,0.5h);
               faceColor = tex2D(ColorMap,irisST.yz).rgb;
          }
          faceColor = lerp(faceColor,LensColor,fadeT);
          hitColor = lerp(missColor,faceColor,smoothstep(0.0h,GRADE,slice));
     }
}
```

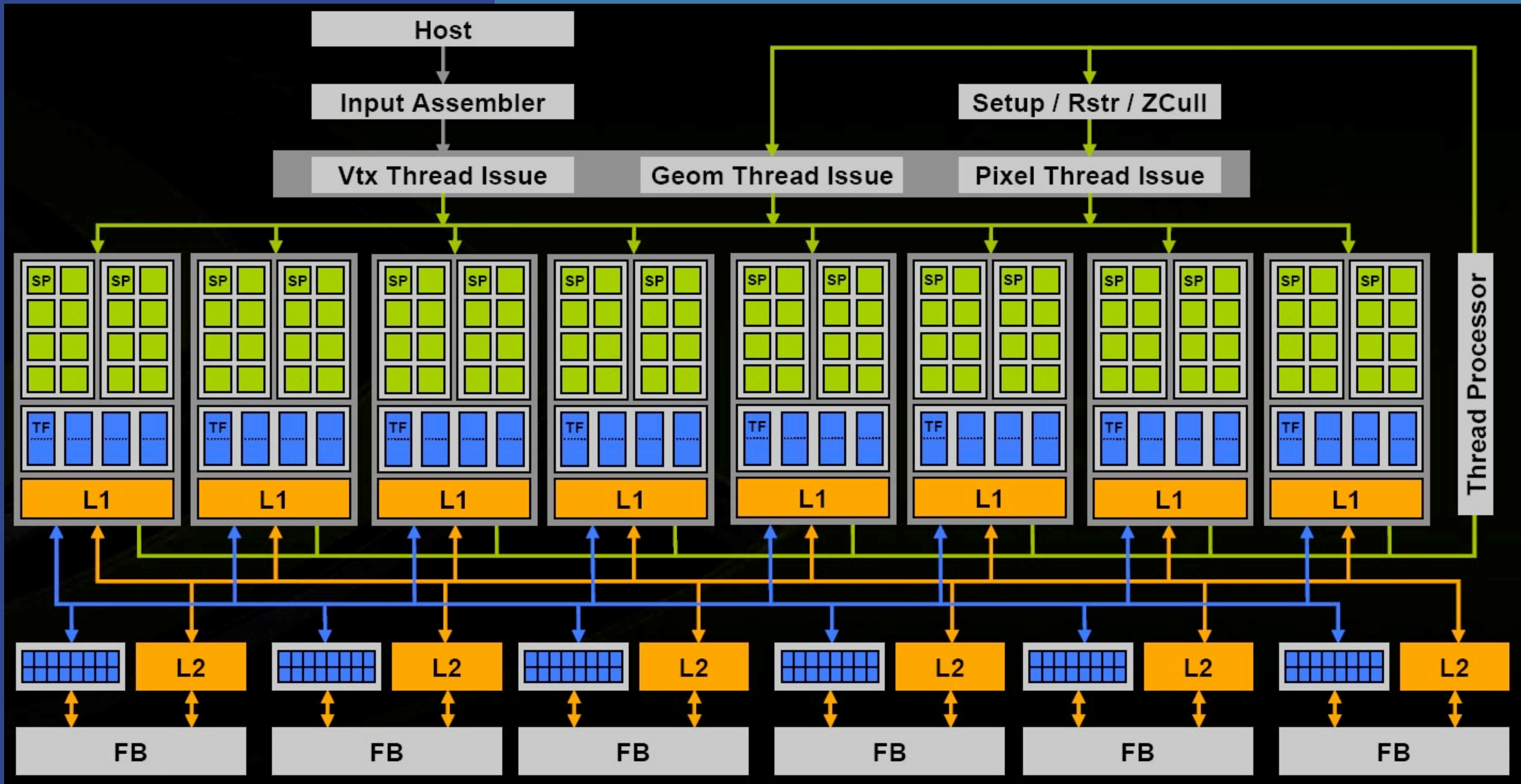


On NVIDIA - series:

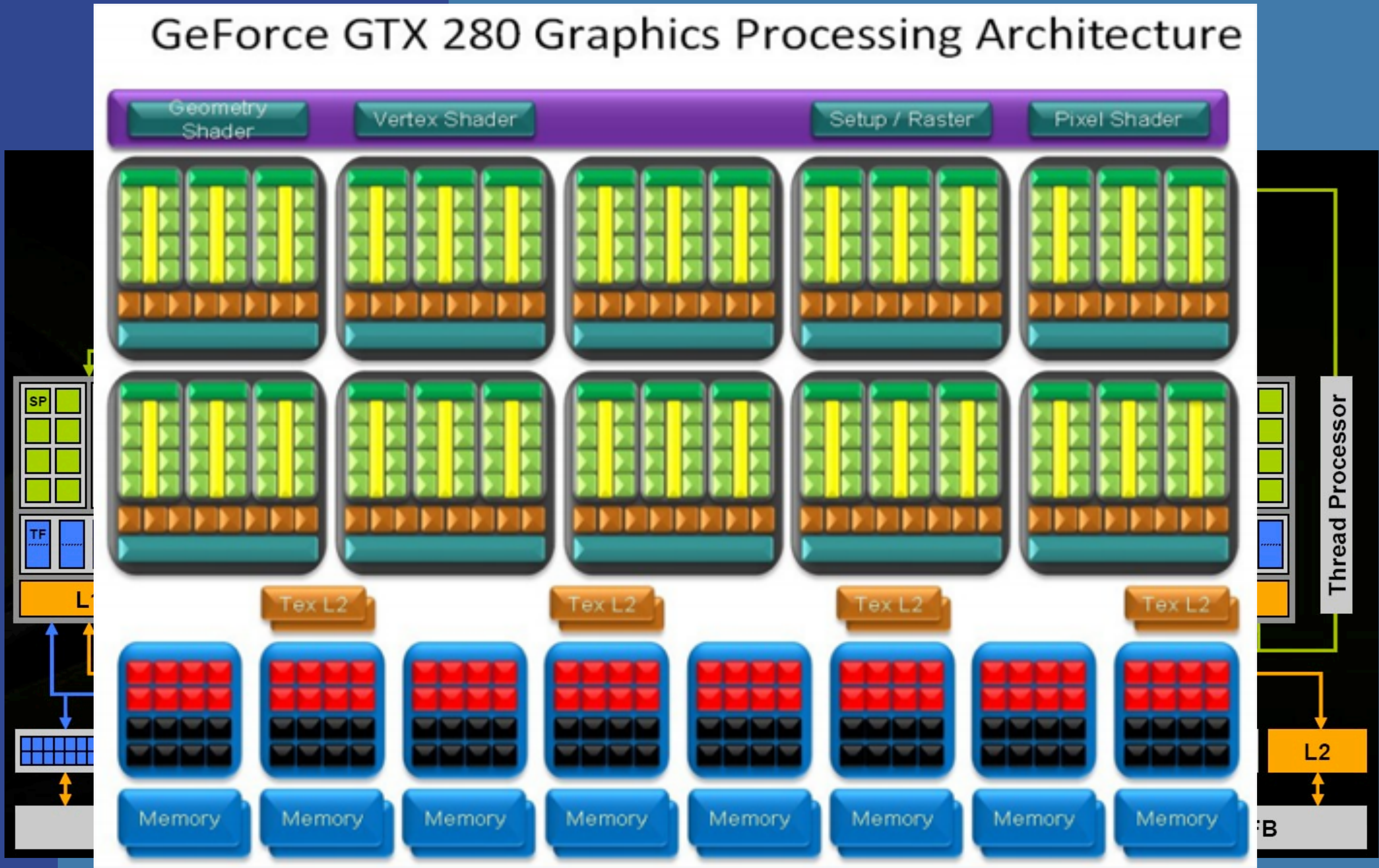
Vertex-, Geometry- and Fragment shaders allocated from a pool of 240 processors



NVIDIA Geforce 8800-architecture



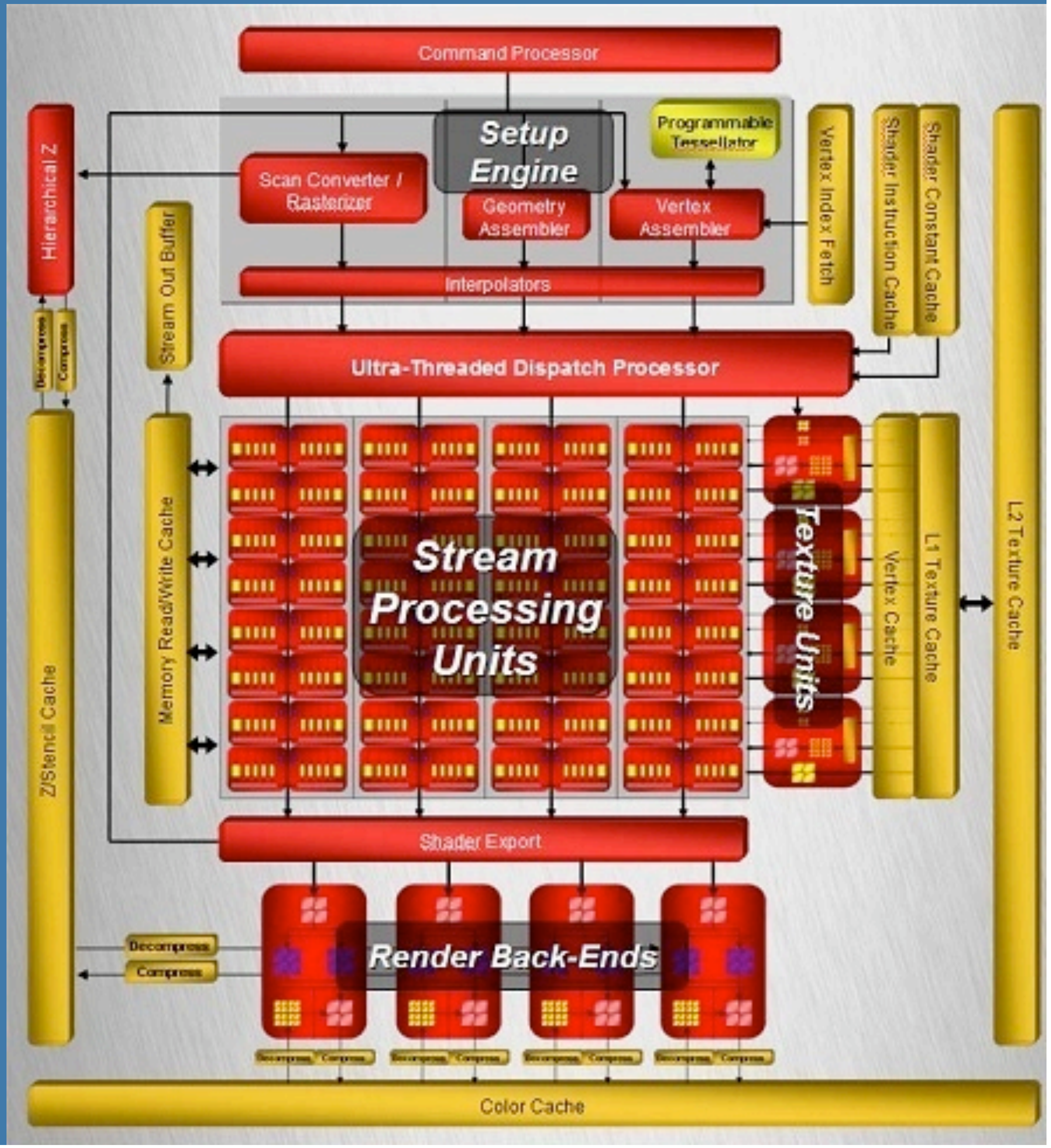
Logic layout



Logic layout

ATI Radeon HD 3000

- 64 cores à 5-float SIMD
  - → 320 stream proc.



Graphics Hardware History

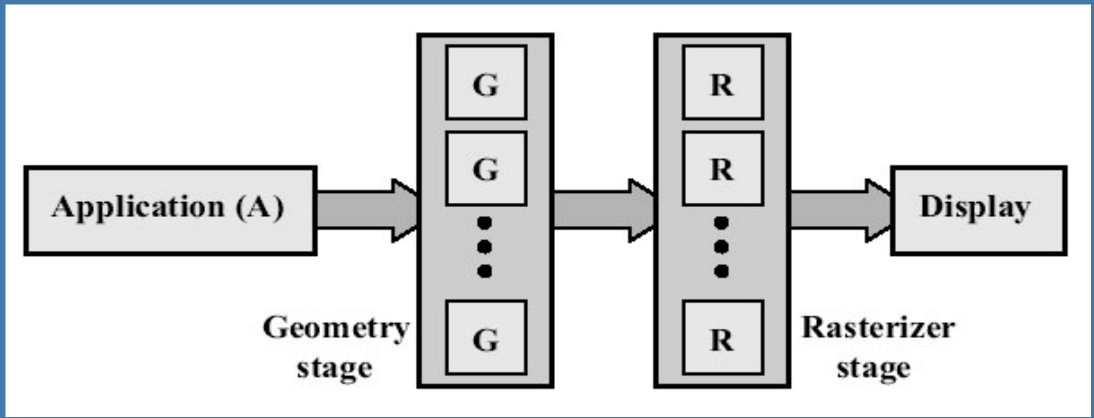
- 80's:
  - linear interpolation of color over a scanline
  - Vector graphics
- 91' Super Nintendo, Neo Geo,
  - Rasterization of 1 single 3D rectangle per frame (FZero)
- 95-96': Playstation 1, 3dfx Voodoo 1
  - Rasterization of whole triangles (triangle setup by Voodoo 2, 1998)
- 99' Geforce (256)
  - Transforms and Lighting (geometry stage)
- 02' 3DLabs WildCat Viper, P10
  - Pixel shaders, integers,
- 02' ATI Radion 9700, GeforceFX
  - Vertex shaders and **Pixel shaders** with floats
- 06' Geforce 8800
  - Geometry shaders, integers and floats, logical operations

Briefly about Graphics HW pipelining

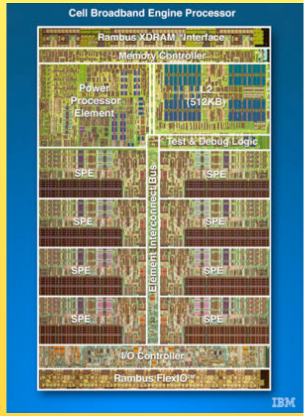
- 2001 In GeForce3: 600-800 pipeline stages!
  - 57 million transistors
  - First Pentium IV: 20 stages, 42 million transistors,
  - Core2 Duo, 271 Mtrans, Intel Core 2 Extreme QX9770 – 820Mtrans.
  - Intel Pentium D 900, 376M trans
- Evolution of cards:
  - 2004 – X800 – 165M transistors
  - 2005 – X1800 – 320M trans, 625 MHz, 750 Mhz mem, 10Gpixels/s, 1.25G verts/s
  - 2004 – GeForce 6800: 222 M transistors, 400 MHz, 400 MHz core/550 MHz mem
  - 2005 – GeForce 7800: 302M trans, 13Gpix/s, 1.1Gverts/s, bw 54GB/s, 430 MHz core,mem 650MHz(1.3GHz)
  - 2006 – GeForce 8800: 681M trans, 39.2Gpix/s, 10.6Gverts/s, bandwidth 103.7 GB/s, 612 MHz core (1500 for shaders), 1080 MHz mem (effective 2160 GHz)
  - 2008 – Geforce 280 GTX: 1.4G trans, 65nm, 602/1296 MHz core, 1107(*2)MHz mem, 142GB/s, 48Gtex/s
- Ghw speed doubles~6 months, CPU speed doubles ~18 months
- Ideally: n stages → n times throughput
  - But latency is high (may also increase)!
  - However, not a problem here
    - Chip runs at about 500 MHz (2ns per clock)
    - 2ns*700=1.4 μs
    - We got about 20 ms per frame (50 frames per second)
- Graphics hardware is simpler to pipeline because:
  - Pixels are (most often) independent of each other
  - Few branches and much fixed functionality
  - Don’t need high clock freq: bandwidth to memory is bottleneck
    - This is changing with increased programmability
  - Simpler to predict memory access pattern (do prefecthing!)

Parallellism

- ”Simple” idea: compute n results in parallel, then combine results
- GeForce 280 GTX: ≤ 240 pixels/clock
  - Many pixels are processed simultaneously
- Not always simple!
  - Try to parallelize a sorting algorithm…
  - But pixels are independent of each other, so simpler for graphics hardware
- Can parallellize both geometry and rasterizer:



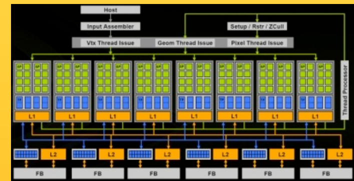
Current and Future Graphics Processors



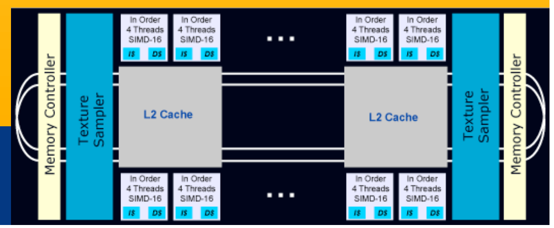
- Cell – 2005
  - 8 cores à 4-float SIMD
  - 256KB L2 cache
  - 128 entry register file
  - 3.2 GHz

PowerXCell 8i Processor – 2008
  - 8 cores à 4-float SIMD
  - 256KB L2 cache
  - 128 entry register file
  - but has better double precission support

- NVIDIA 8800 GTX – Nov 2006
  - 16 cores à 8-float SIMD (GTX 280 – 30 cores à 8-float SIMD, june ´08
  - 16 KB L1 cache, 64KB L2 cache (rumour)
  - 1.2-1.625 GHz



- Larrabee - 2009
  - 16-24 cores à 16-float SIMD
  - Core = 16-float SIMD (=512bit FPU) + x86 proc with loops, branches + scalar ops, 4 threads/core
  - 32KB L1cache, 256KB L2-cache
  - 1.7-2.4 GHz

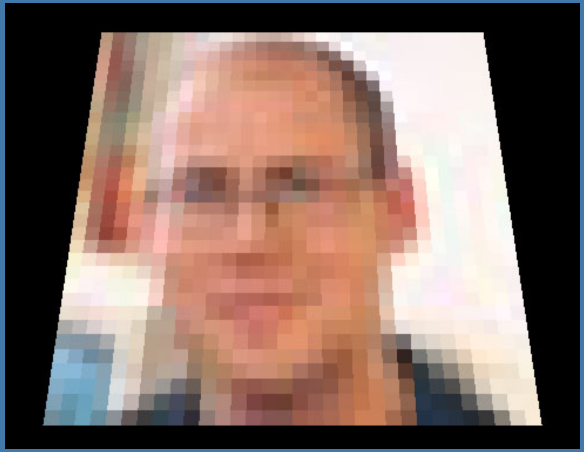


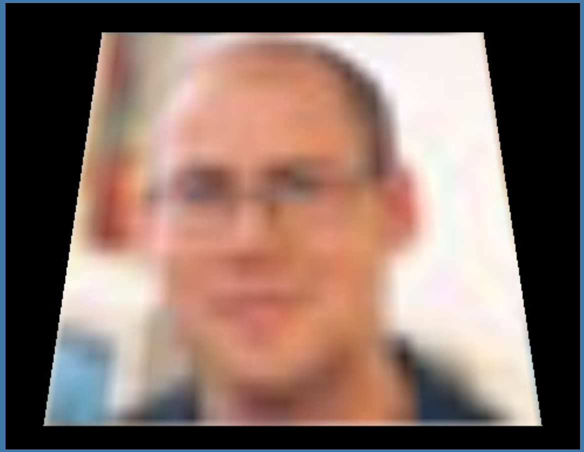
Memory bandwith usage is huge!!

Mainly due to texture reads

FILTERING:

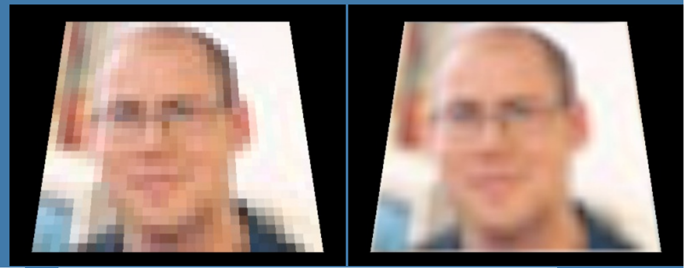
- For magnification: Nearest or Linear (box vs Tent filter)



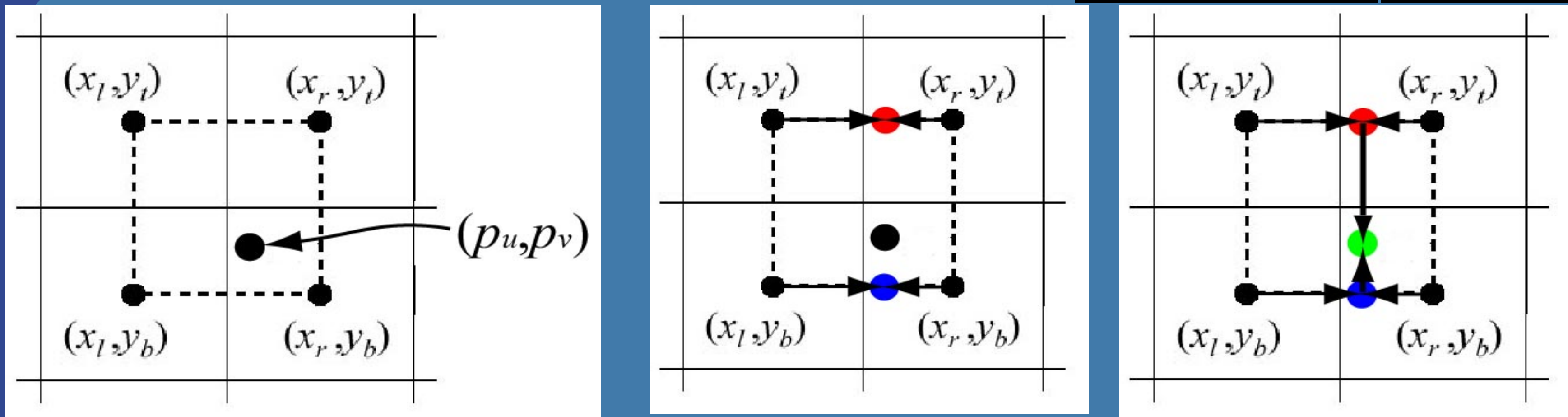


- For minification:
  - Bilinear – using mipmapping
  - Trilinear – using mipmapping
  - Anisotropic – some mipmap lookups along line of anisotropy

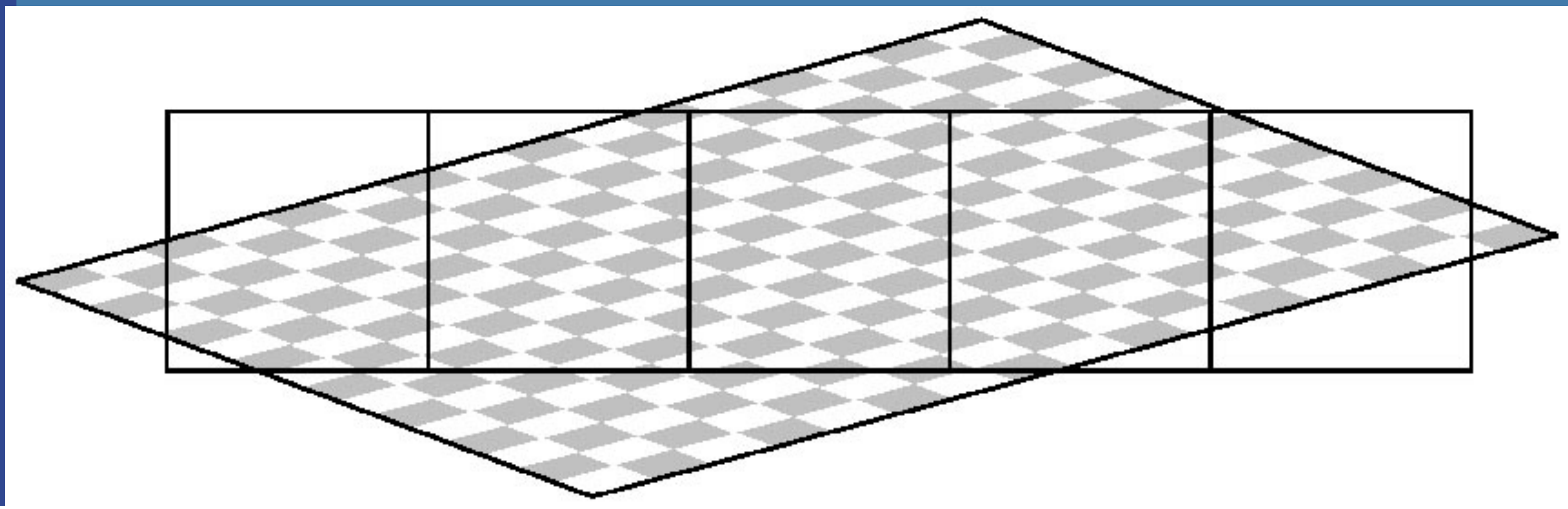
Interpolation



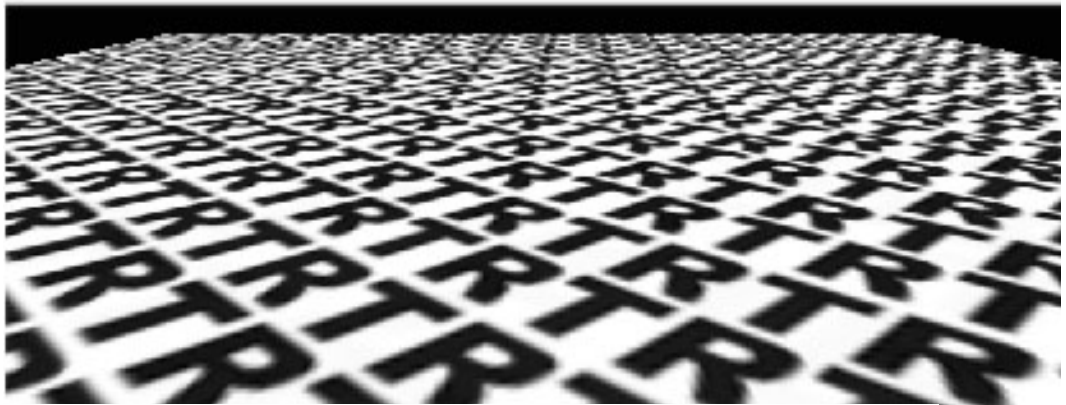
Magnification

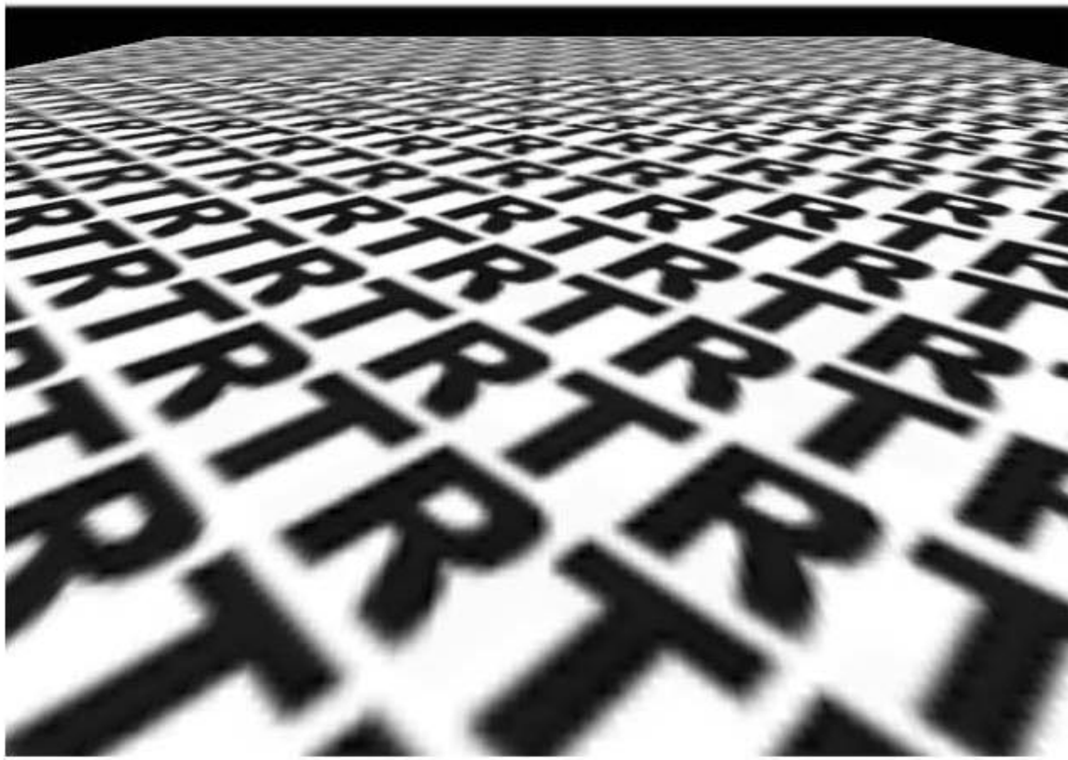


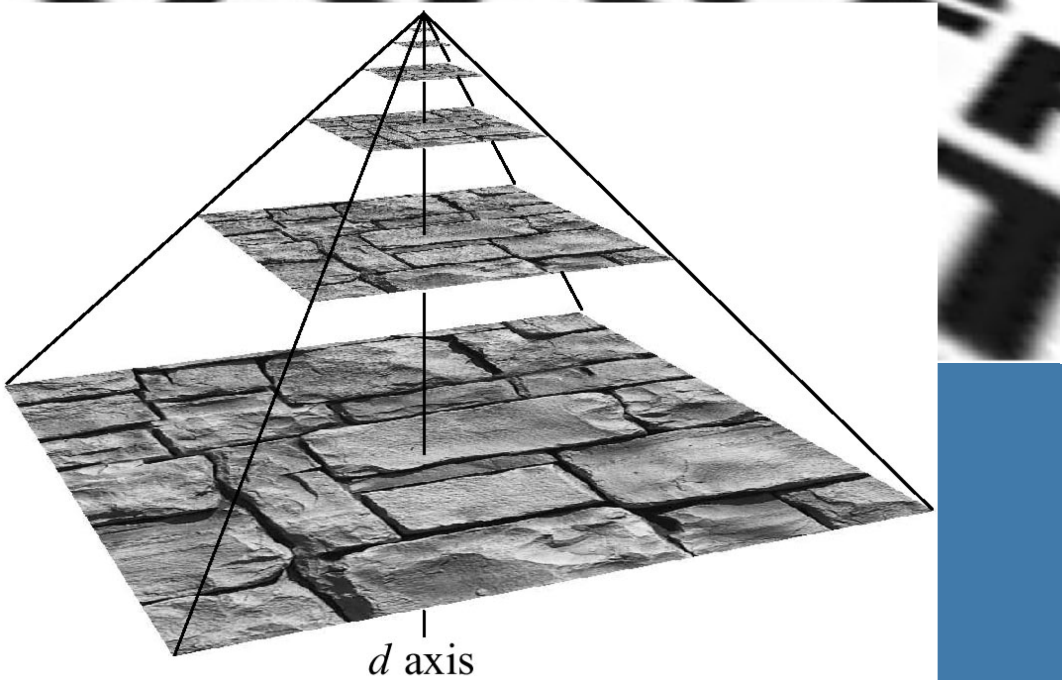
Minification



Bilinear filtering using Mipmapping





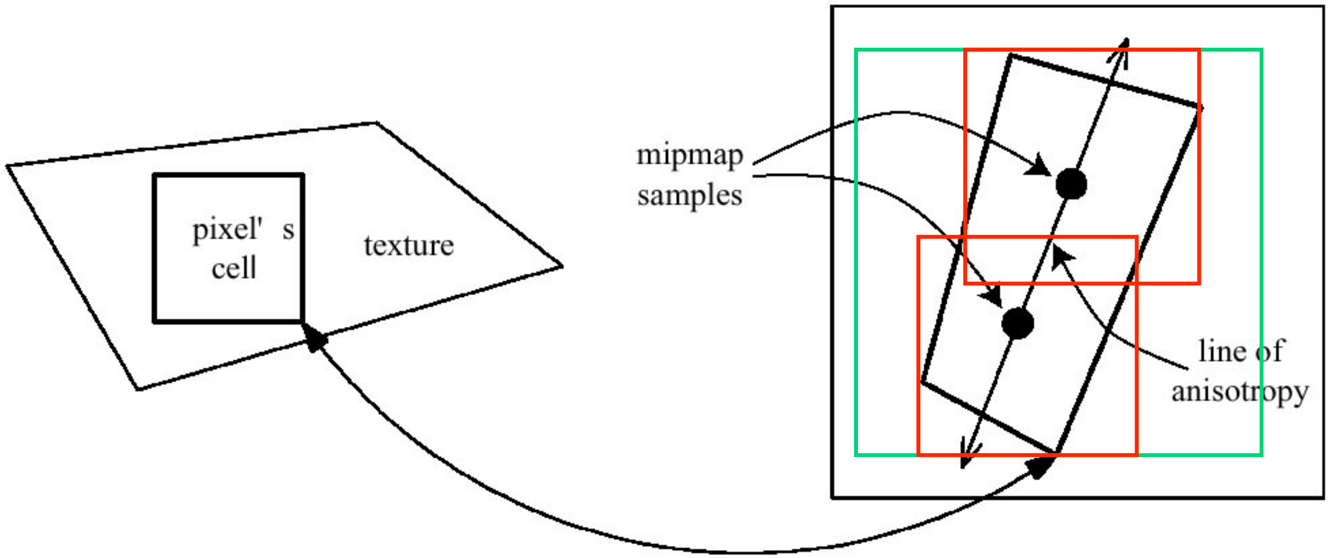


Anisotropic texture filtering

And we haven't even used float-textures yet...

nor 3D textures…

pixel space | texture space



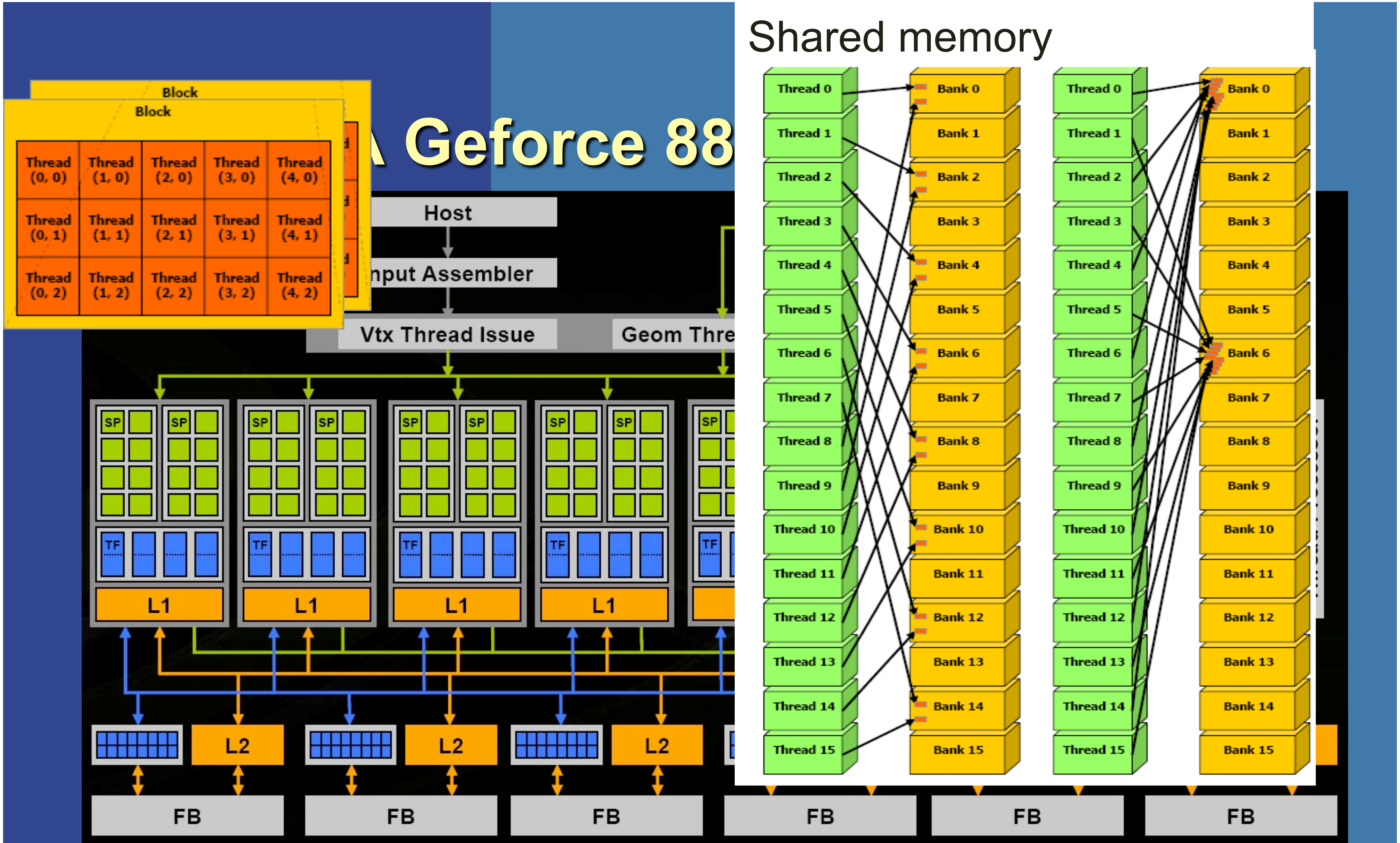
Wish list:

1 sample = 32 bytes (or 512 for 16x ani. filter.)

240 proc * 500MHz * 32 bytes = 3840 GB/s per texture (60K GB/s)

Memory bandwidth usage is huge!!

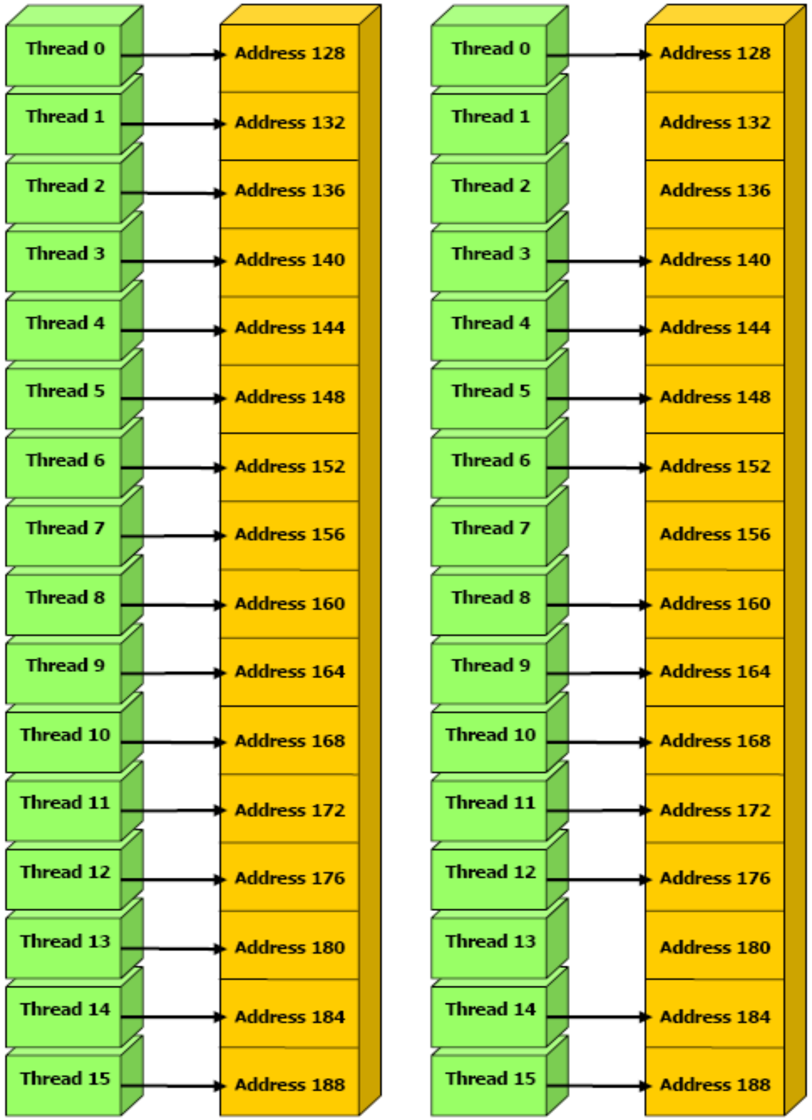
- Assume GDDR3 (2x faster than DDRAM) at 2214 MHz, 512 bits per access: => 141.7 Gb/s
- On top of that bandwith usage is never 100%, and Multiple textures, anti-aliasing (supersampling), will use up alot more bandwidth
- However, there are many techniques to reduce bandwith usage:
  - Texture caching with prefetching
  - Texture compression
  - Z-compression
  - Z-occlusion testing (HyperZ)



Logic layout

Global Memory

- Coalesced reads and writes



With courtesy of DICE: RalliSport Challenge 2

Va e de för bra me datorgrafik då ?

With courtesy of Malin Grön

Image from Surgical Science

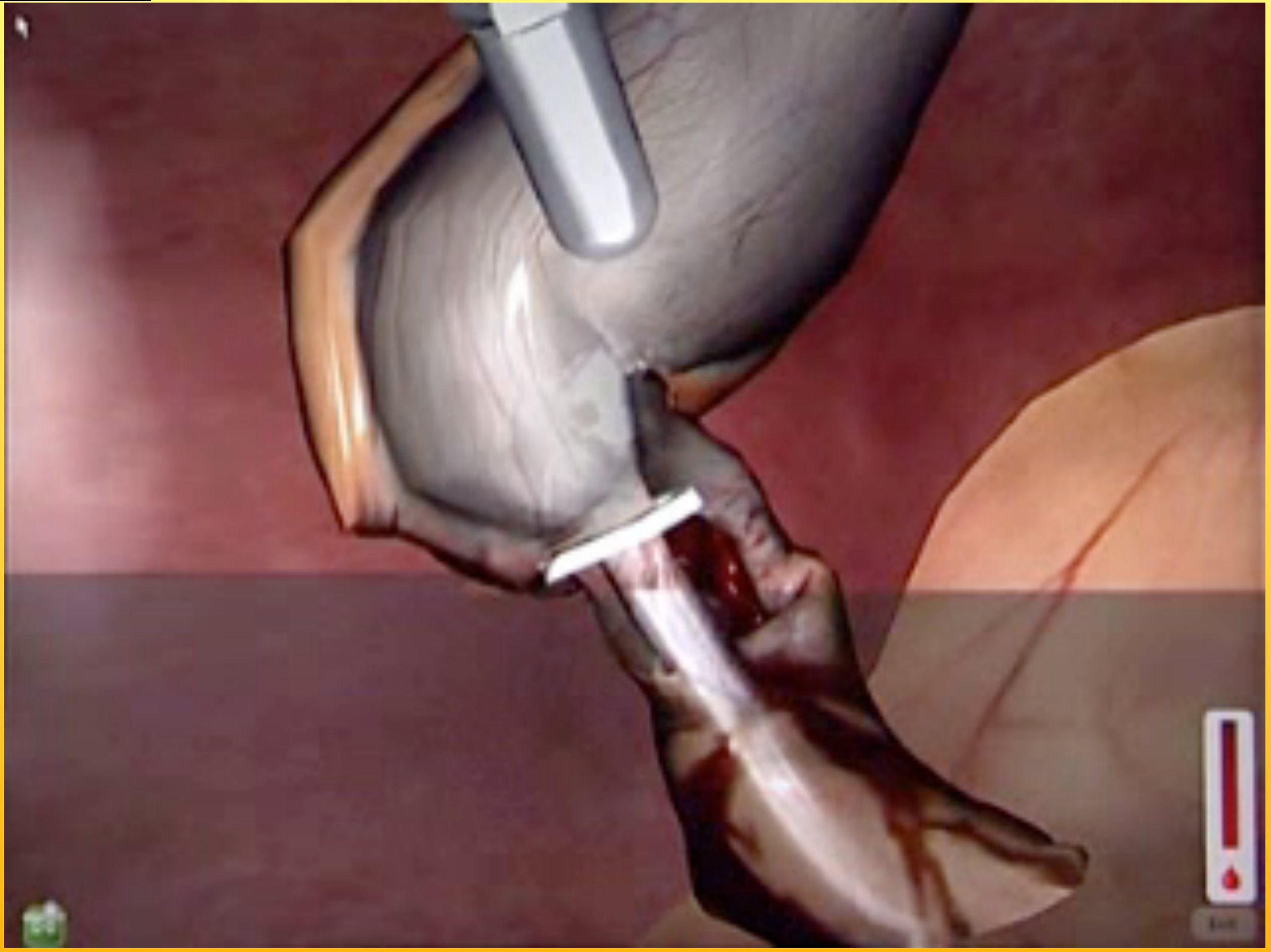


Image from Surgical Science

CHALMERS

Vill du veta mer?

Välkommen till TDA361
Computer Graphics

Lp1, 2009