Programmable graphics pipeline

Adapted from

Suresh Venkatasubramanian UPenn

The evolution of the pipeline

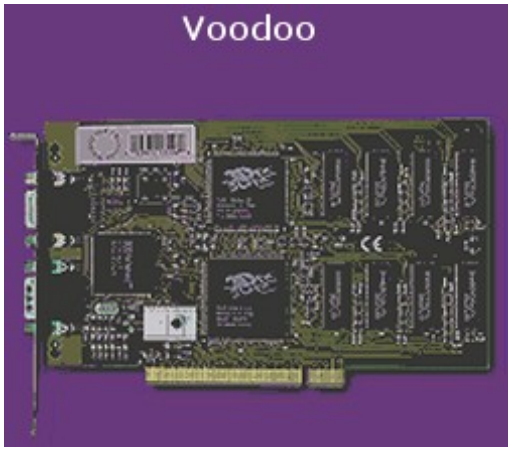
Elements of the graphics pipeline:

1. A scene description: vertices, triangles, colors, lighting
2. Transformations that map the scene to a camera viewpoint
3. “Effects”: texturing, shadow mapping, lighting calculations
4. Rasterizing: converting geometry into pixels
5. Pixel processing: depth tests, stencil tests, and other per-pixel operations.

Parameters controlling design of the pipeline:

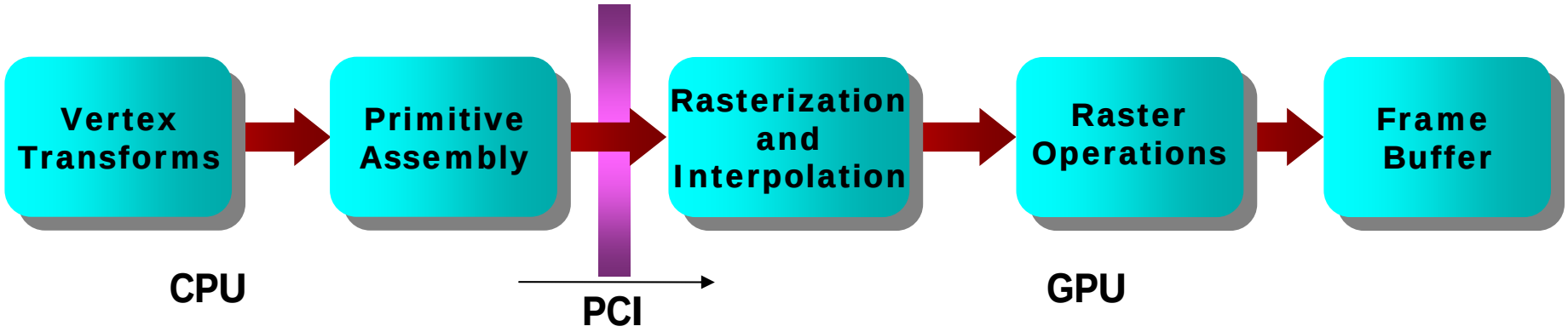
1. Where is the boundary between CPU and GPU ?
2. What transfer method is used ?
3. What resources are provided at each step ?
4. What units can access which GPU memory elements ?

Generation I: 3dfx Voodoo (1996)



http://accelenation.com/?ac.id.123.2

- One of the first true 3D game cards
- Worked by supplementing standard 2D video card.
- Did not do vertex transformations: these were done in the CPU
- Did do texture mapping, z-buffering.

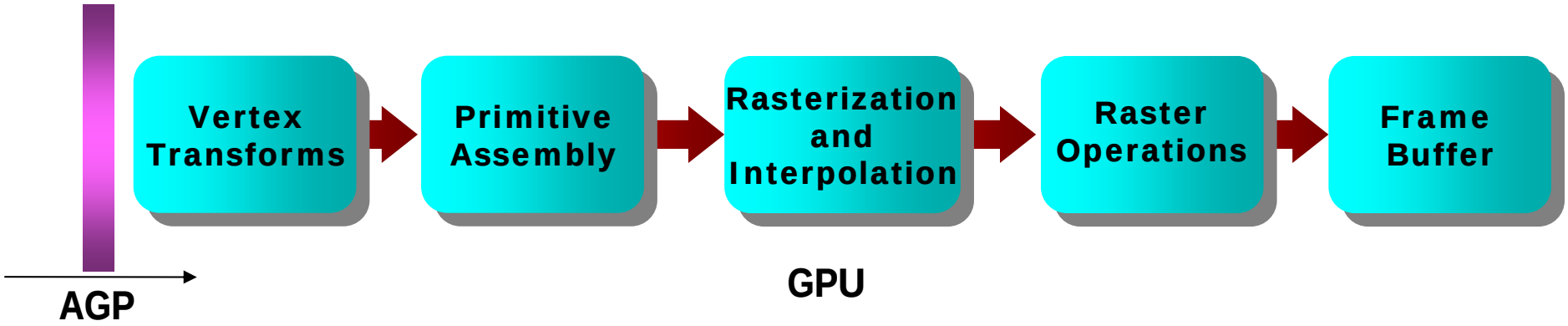


Generation II: GeForce/Radeon 7500 (1998)

GeForce 256

http://accelenation.com/?ac.id.123.5

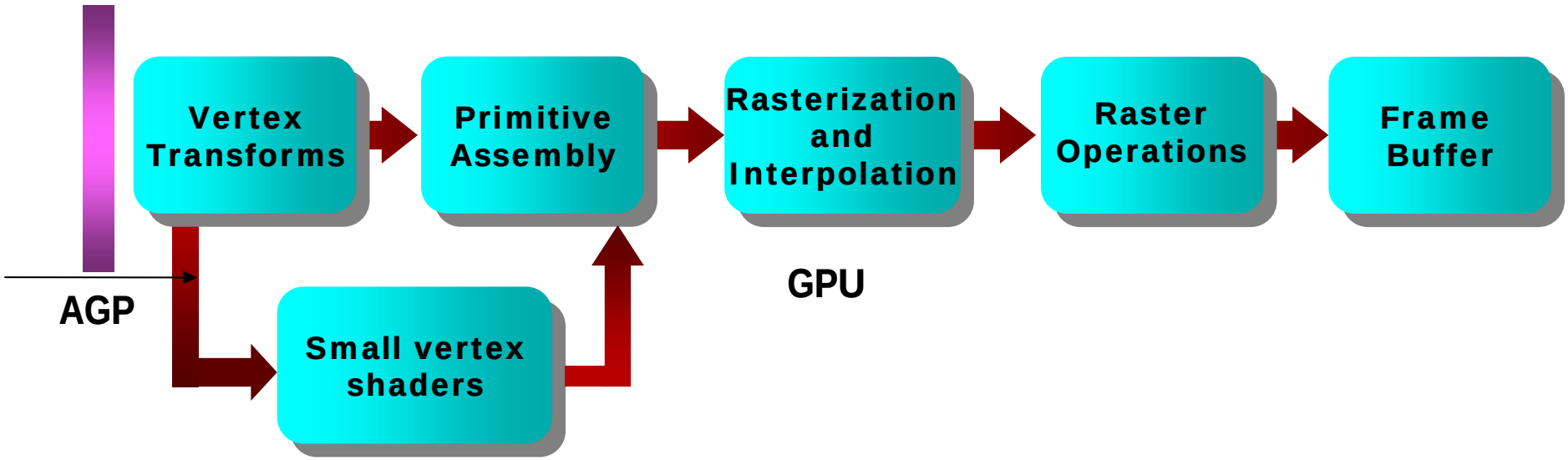
- Main innovation: shifting the transformation and lighting calculations to the GPU
- Allowed multi-texturing: giving bump maps, light maps, and others..
- Faster AGP bus instead of PCI



Generation III: GeForce3/Radeon 8500(2001)

http://accelenation.com/?ac.id.123.7

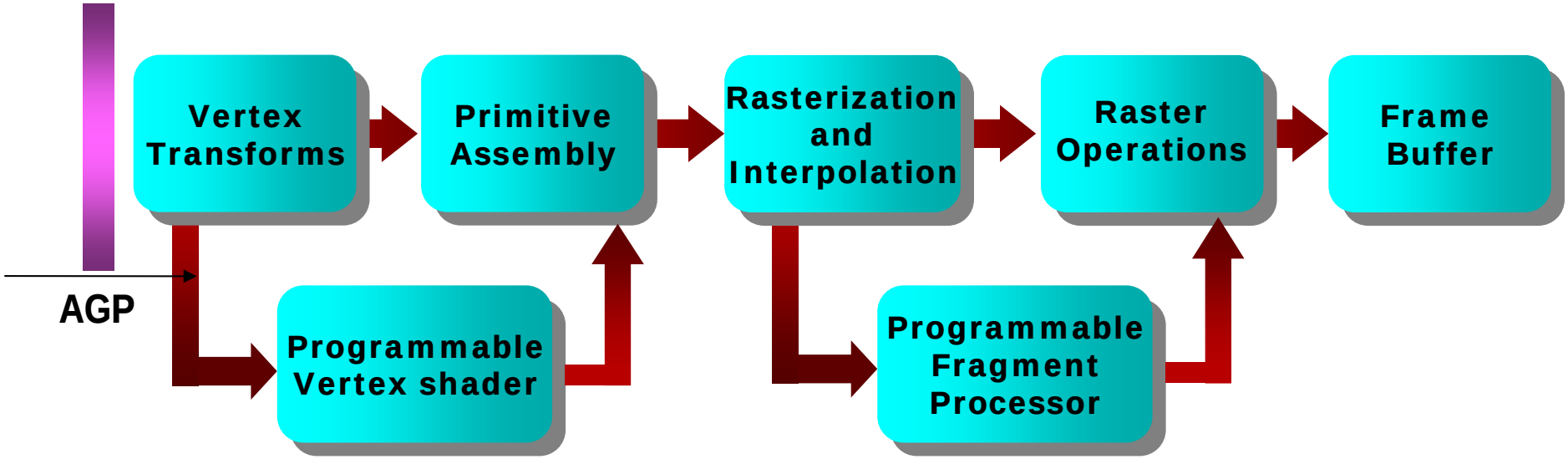
- For the first time, allowed limited amount of programmability in the vertex pipeline
- Also allowed volume texturing and multi-sampling (for antialiasing)



Generation IV: Radeon 9700/GeForce FX (2002)

http://accelenation.com/?ac.id.123.8

- This generation is the first generation of fully-programmable graphics cards
- Different versions have different resource limits on fragment/vertex programs



Generation IV.V: GeForce6/X800 (2004)

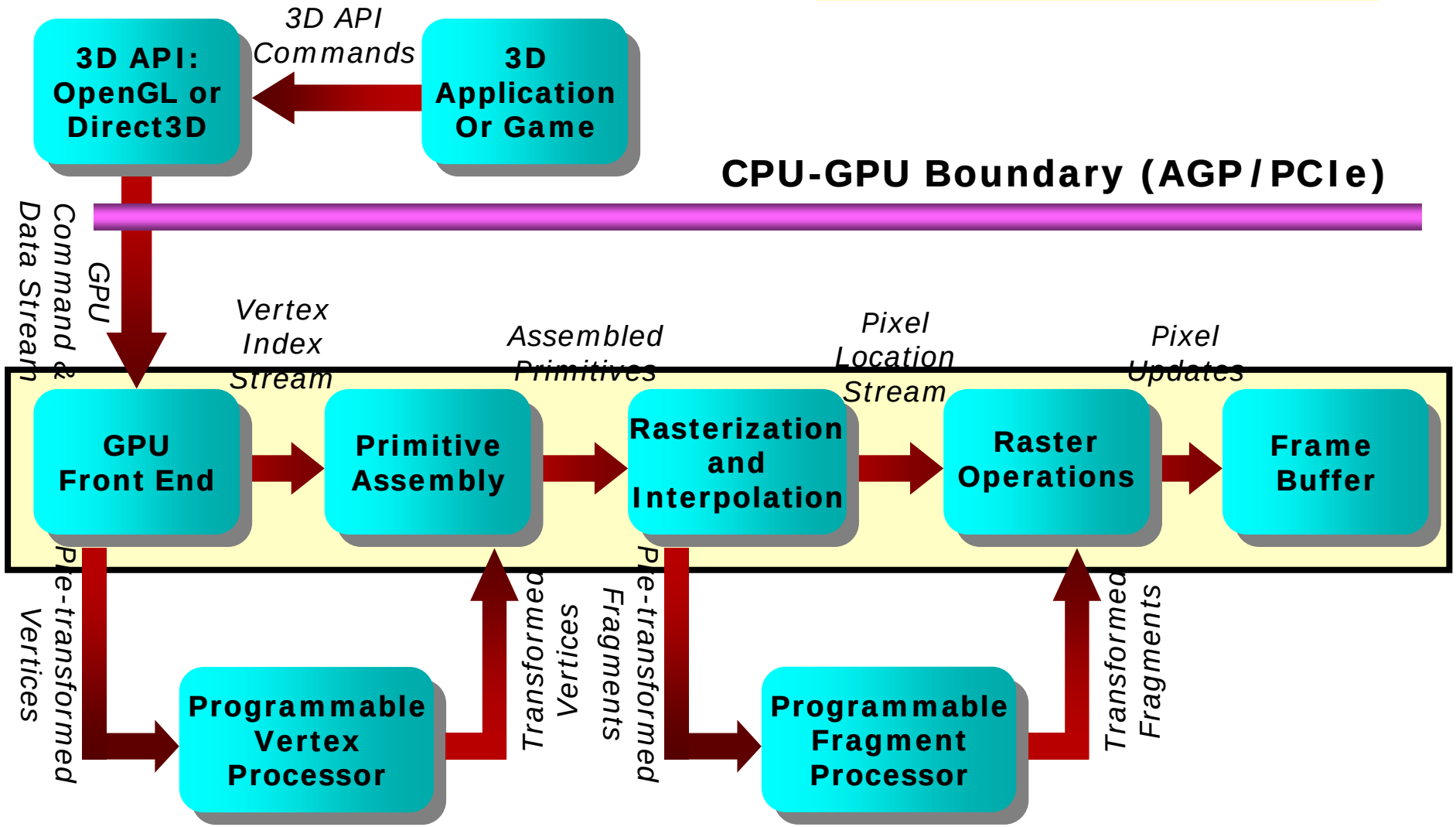
Not exactly a quantum leap, but…

- Simultaneous rendering to multiple buffers
- True conditionals and loops
- Higher precision throughput in the pipeline (64 bits end-to-end, compared to 32 bits earlier.)
- PCIe bus
- More memory/program length/texture accesses

New Generation: CUDA GeForce8800/Telsa (2007)

- “Compute Unified Device Architecture”
- General purpose programming model
  - User kicks off batches of threads on the GPU
  - GPU = dedicated super-threaded, massively data parallel co-processor
- Targeted software stack
  - Compute oriented drivers, language, and tools
- Driver for loading computation programs into GPU
  - Standalone Driver - Optimized for computation
  - Interface designed for compute - graphics free API
  - Data sharing with OpenGL buffer objects
  - Guaranteed maximum download & readback speeds
  - Explicit GPU memory management

Fixed-function pipeline



A closer look at the fixed-function pipeline

Pipeline Input

Vertex

(x, y, z)

(r, g, b,a)

(Nx,Ny,Nz)

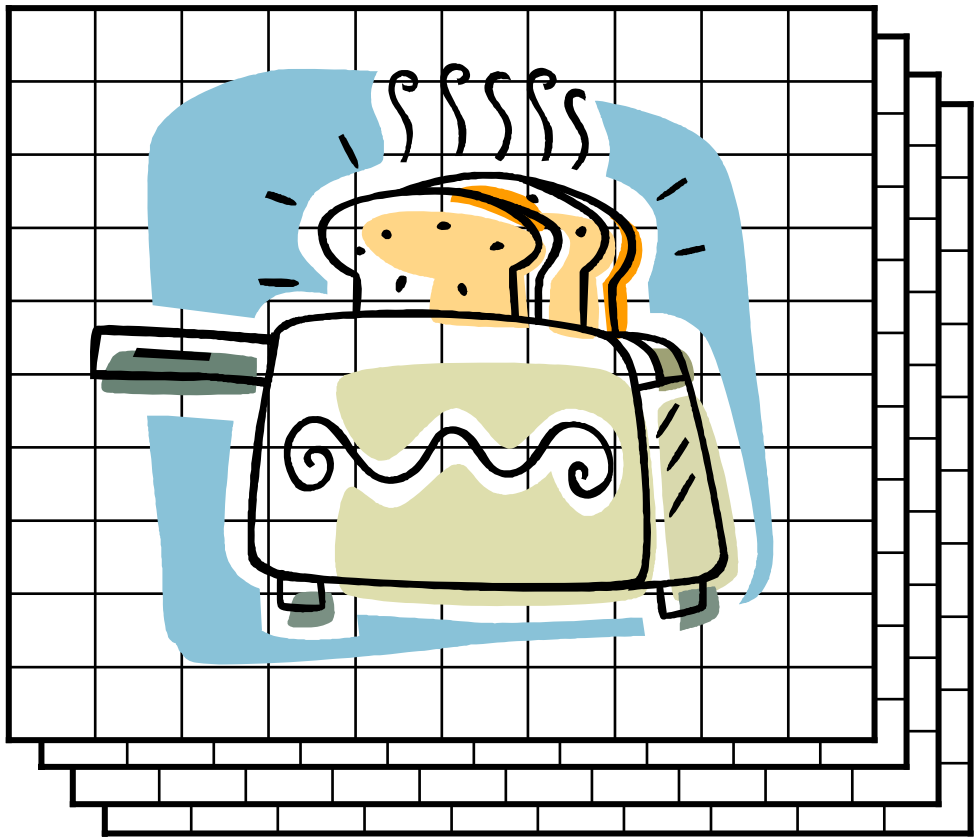
(tx, ty,[tz])

(tx, ty)

(tx, ty)

Material properties*

Image F(x,y) = (r,g,b,a)



ModelView Transformation

- Vertices mapped from object space to world space
- M = model transformation (scene)
- V = view transformation (camera)

$$\begin{pmatrix} X' \\ Y' \\ Z' \\ W' \end{pmatrix} = M * V * \begin{pmatrix} X \\ Y \\ Z \\ 1 \end{pmatrix}$$

Each matrix transform is applied to each vertex in the input stream. Think of this as a kernel operator.

Lighting

Lighting information is combined with normals and other parameters at each vertex in order to create new colors.

$$\text{Color}(v) = \textit{emissive} + \textit{ambient} + \textit{diffuse} + \textit{specular}$$

Each term in the right hand side is a function of the vertex color, position, normal and material properties.

Clipping/Projection/Viewport(3D)

- More matrix transformations that operate on a vertex to transform it into the viewport space.
- Note that a vertex may be eliminated from the input stream (if it is clipped).
- The viewport is two-dimensional: however, vertex z-value is retained for depth testing.

Rasterizing+Interpolation

- All primitives are now converted to fragments.
- Data type change ! Vertices to fragments

Fragment attributes:

(r,g,b,a)

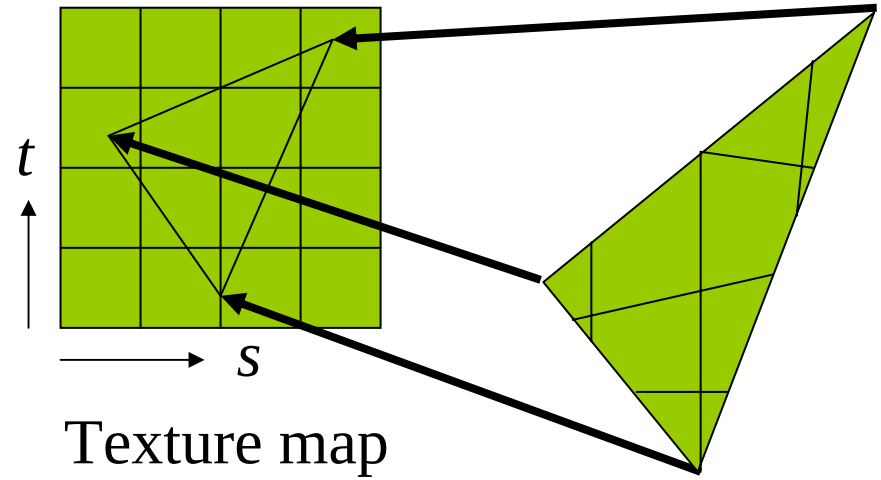
(x,y,z,w)

(tx,ty), …

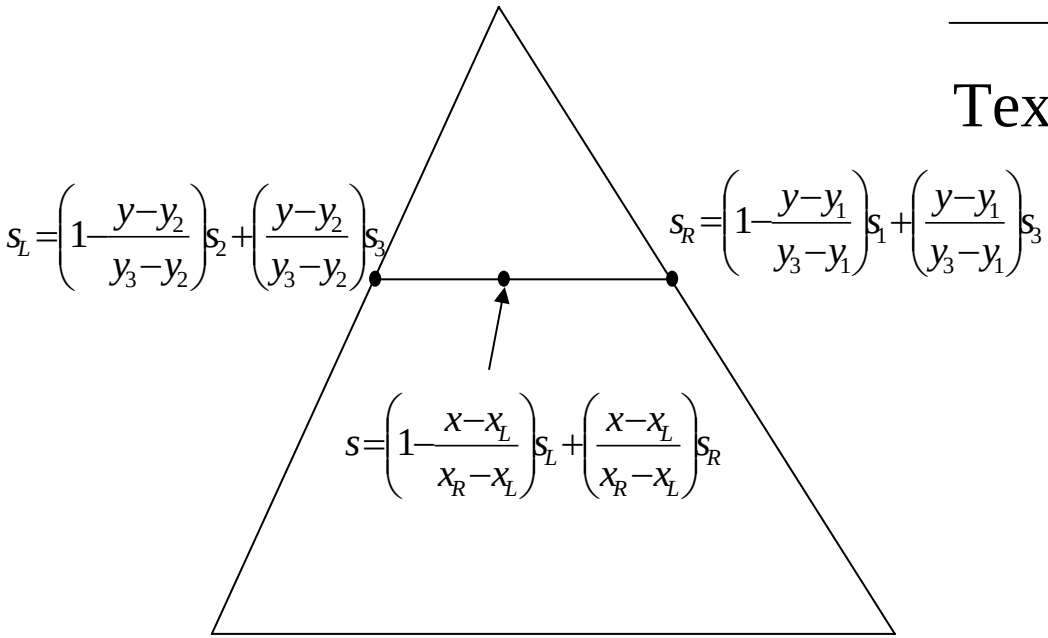
Texture coordinates are interpolated from texture coordinates of vertices.

This gives us a linear interpolation operator for free. VERY USEFUL !

Texture Interpolation



Texture map



Triangle in world space

Per-fragment operations

- The rasterizer produces a stream of fragments.
- Each fragment undergoes a series of tests with increasing complexity.

Test 1: Scissor

If (fragment lies in fixed rectangle) let it pass **else** discard it

Test 2: Alpha

If(fragment.a >= <constant>) let it pass **else** discard it.

Per-fragment operations

- Stencil test: S(x, y) is stencil buffer value for fragment with coordinates (x,y)
- If f(S(x,y)), let pixel pass else kill it. Update S(x, y) conditionally depending on f(S(x,y)) and g(D(x,y)).
- Depth test: D(x, y) is depth buffer value.
- If g(D(x,y)) let pixel pass else kill it. Update D(x,y) conditionally.

Per-fragment operations

- Stencil and depth tests are the only tests that can change the state of internal storage (stencil buffer, depth buffer). This is very important.
- Unfortunately, stencil and depth buffers have lower precision (8, 24 bits resp.)

Post-processing

- Blending: pixels are accumulated into final framebuffer storage

  new-val = old-val *op* pixel-value

  If *op* is +, we can sum all the (say) red components of pixels that pass all tests.

Problem: In generation<= IV, blending can only be done in 8-bit channels (the channels sent to the video card); precision is limited.