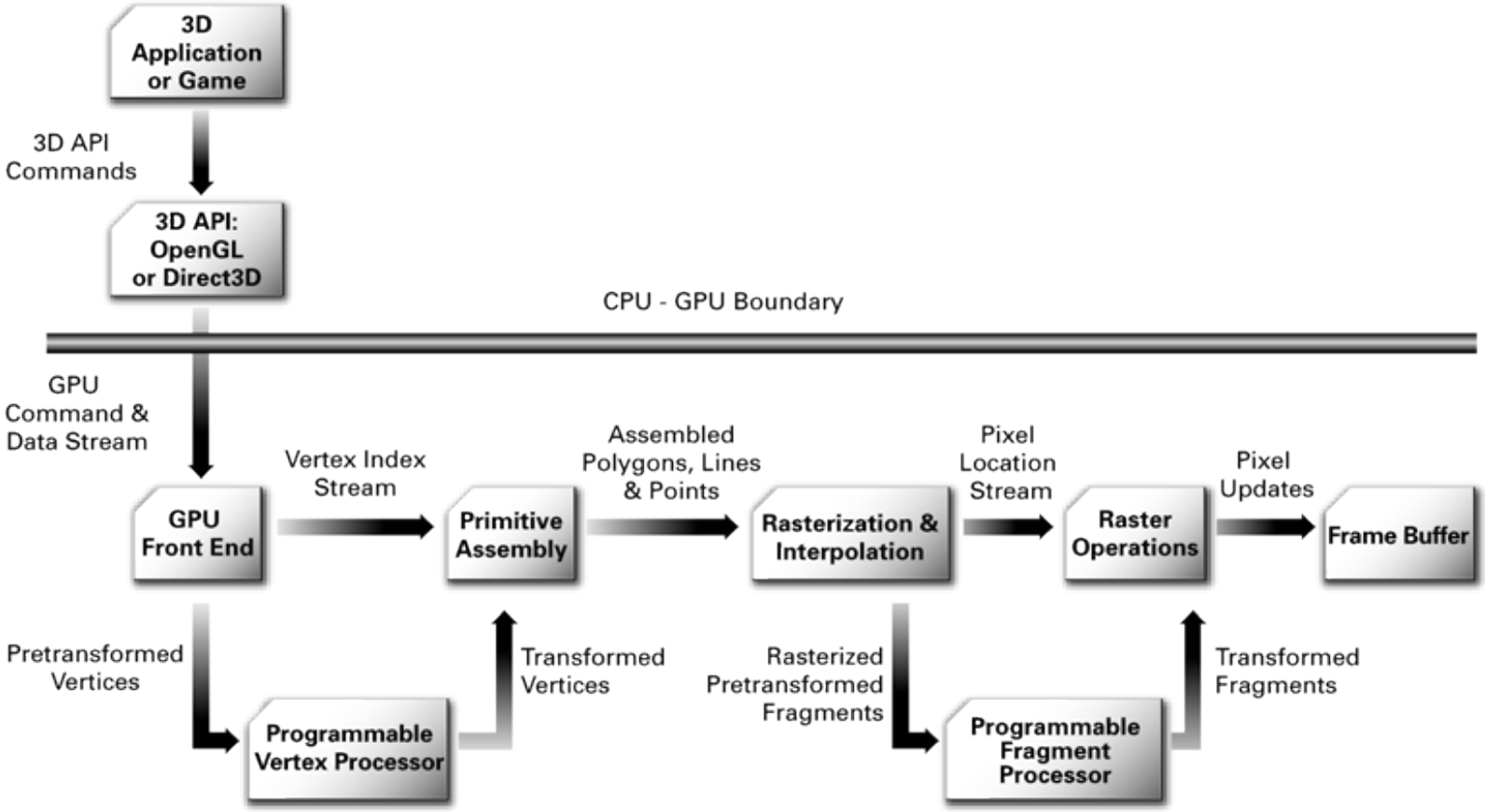
Game Architecture

2/26/16: Shaders


NVIDIA.

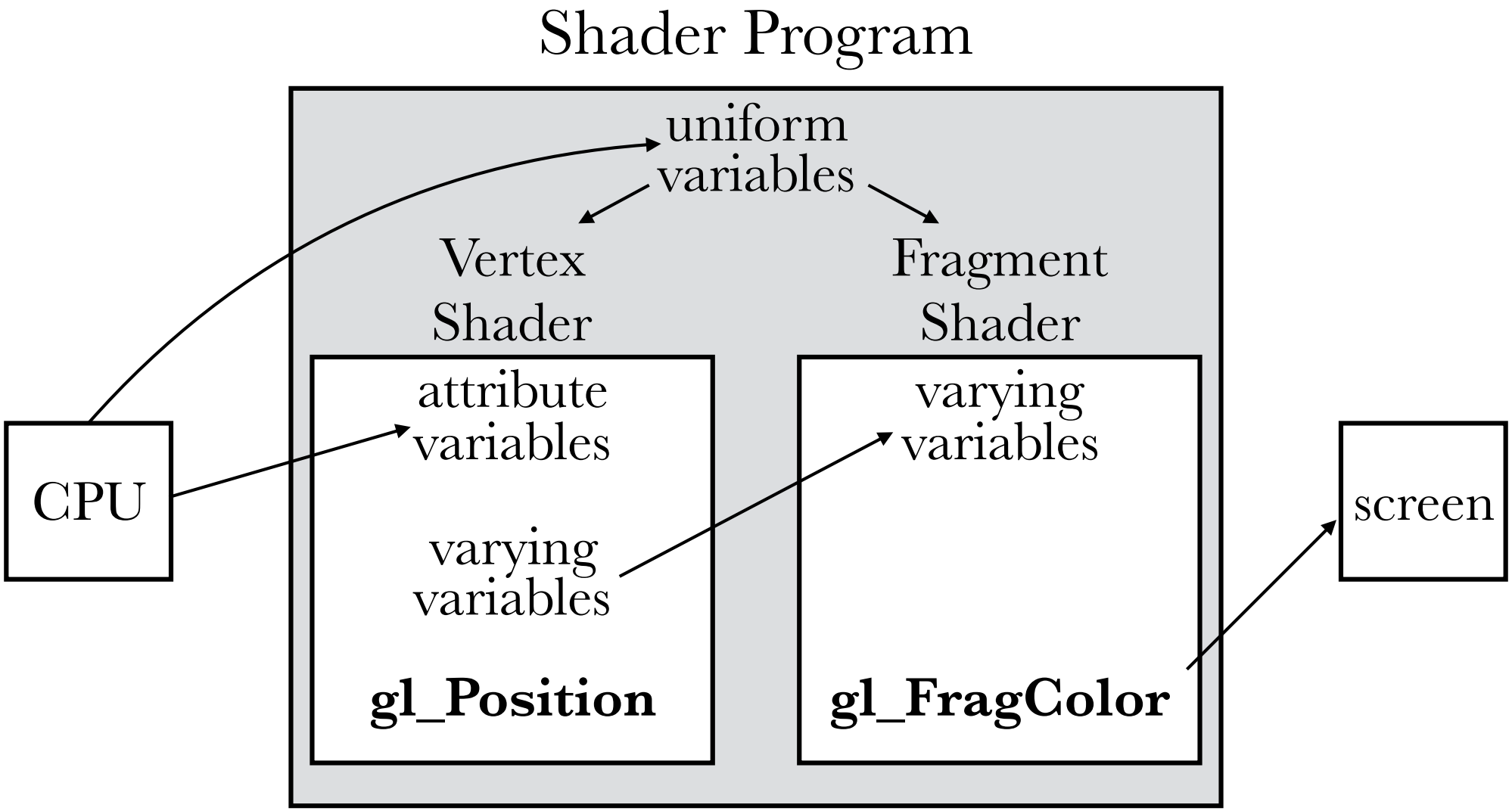

3D Application or Game
3D API Commands
3D API: OpenGL or Direct3D
CPU - GPU Boundary
GPU Command & Data Stream
GPU Front End
Vertex Index Stream
Primitive Assembly
Assembled Polygons, Lines & Points
Rasterization & Interpolation
Pixel Location Stream
Raster Operations
Pixel Updates
Frame Buffer
Pretransformed Vertices
Programmable Vertex Processor
Transformed Vertices
Rasterized Pretransformed Fragments
Programmable Fragment Processor
Transformed Fragments

OpenGL Shading

- Vertex Processor
  - A programmable unit that operates on incoming vertices and their associated data. Operates on vertices individually
- Fragment Processor
  - Operates on pixels and their associated data. Cannot change a fragment's (x, y) position. Updates frame-buffer memory or texture memory

Storage Qualifiers

- `attribute` – linkage between a vertex shader and WebGL for per-vertex data
- `varying` – linkage between a vertex shader and fragment shader for interpolated data
- `uniform` – value does not change across the primitive being processed. Forms the linkage between a shader, WebGL, and the app


Shader Program
uniform
variables
Vertex
Shader
Fragment
Shader
attribute
variables
varying
variables
varying
variables
gl_Position
gl_FragColor
CPU
screen

Types

void	mat4	bvec2
float	vec2	bvec3
int	vec3	bvec4
void	vec4	sampler2D
bool	ivec2	struct
mat2	ivec3	float x[5];
mat3	ivec4	

Vector Components

- For notational convenience, but letters must come from same set:
- {x,y,z,w} used for position or normals
- {r,g,b,a} used for color
- {s,t,p,q} used for textures

Matrix Components

- `mat4 m;`
- `m[1] = vec4(2.0); //` sets the second column to all 2.0
- `m[0][0] = 1.0; //` sets the upper-left element to 1.0
- `m[2][3] = 2.0; //` sets the fourth element of the third column to 2.0

The Burning Question of Vector Representation

$$\mathbf{v} = \begin{vmatrix} a_1 \\ a_2 \\ a_3 \end{vmatrix} \quad ? \qquad \text{or} \qquad \mathbf{v} = \begin{vmatrix} a_1 & a_2 & a_3 \end{vmatrix} \; ?$$

$$\mathbf{v}' = (\mathbf{C}(\mathbf{B}(\mathbf{A}\mathbf{v}))) \qquad\qquad \mathbf{v}' = (((\mathbf{v}\mathbf{A})\mathbf{B})\mathbf{C})$$

Structure

- statements and declarations
- function definitions
- selection (`if-else` and `switch-case-default`)
- iteration (`for`, `while`, and `do-while`)
- jumps (`discard`, `return`, `break`, and `continue`)

Built-in Functions

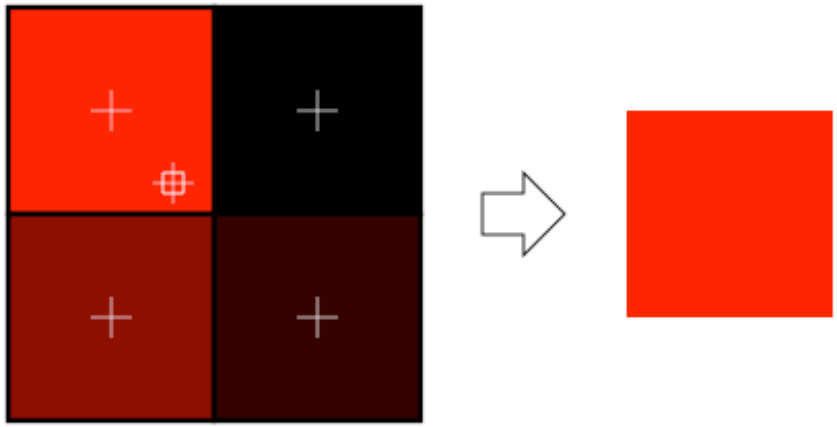
- `radians(deg)`
- `degrees(rad)`
- `sin(angle)`
- `cos(angle)`
- `tan(angle)`
- `asin(x)`
- `acos(x)`
- `atan(y,x)`
- `atan(y_over_x)`
- `pow(x,y)`
- `exp(x)`
- `log(x)`
- `exp2(x)`
- `log2(x)`
- `sqrt(x)`
- `inversesqrt(x)`

Built-in Functions

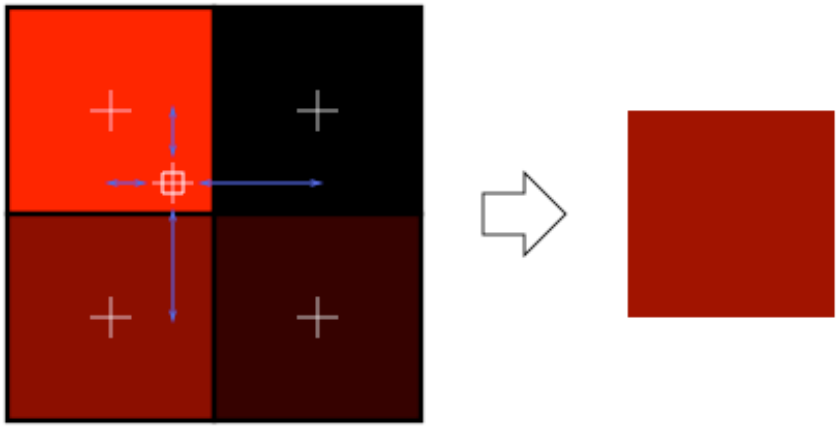
- `abs(x)`
- `sign(x)`
- `floor(x)`
- `ceil(x)`
- `fract(x)`
- `mod(x, y)`
- `min(x, y)`
- `max(x, y)`
- `clamp(x, minVal, maxVal)`
- `mix(x, y, a)`
- `step(edge, x)`
- `smoothstep(edge0, edge1, x)`
- `length(v)`
- `distance(v0, v1)`
- `dot(v0, v1)`
- `cross(v0, v1)`
- `normalize(v)`
- `reflect(i, n)`
- `refract(i, n, eta)`

Texture Lookup Functions

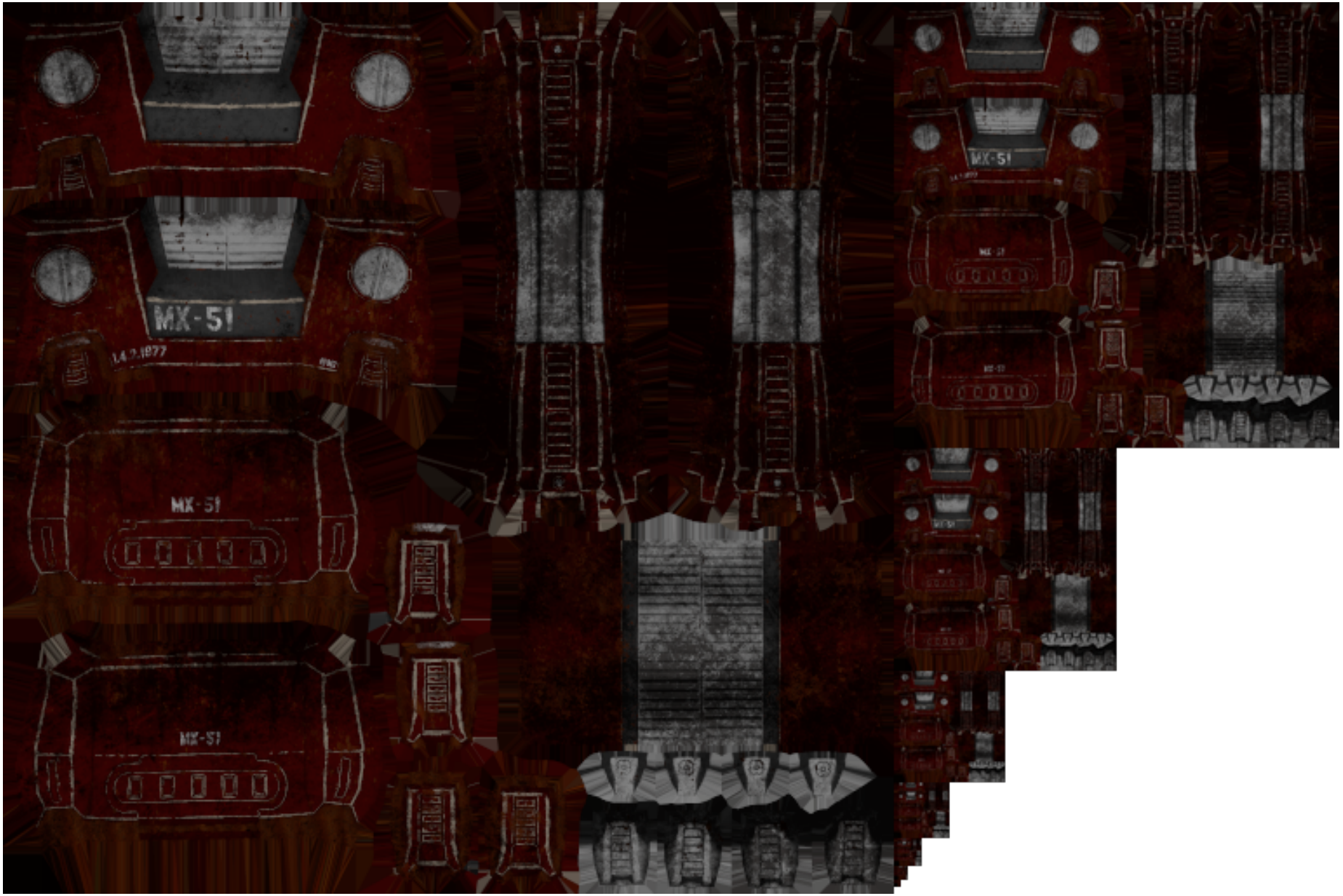
- Several, but mostly:
- `vec4 texture2D(sampler2D, vec2 P)`



GL_NEAREST



GL_LINEAR


MX-51
14.2.1877
MX-51
MX-51
MX-51