GPGPU Programming

Shih-hsuan (Vincent) Hsu
Communication and Multimedia Laboratory
CSIE, NTU

Outline

- Why GPGPU?
- Programmable Graphics Hardware
- Programming Systems
- Writing GPGPU Programs
- Examples
- References

Why GPGPU?

- GPGPU
  - General-Purpose computation on GPU
  - GPU: Graphics Processing Unit
- GPU is probably today's most powerful computational hardware for the dollar
- Advancing at incredible rates
  - # of transistors:

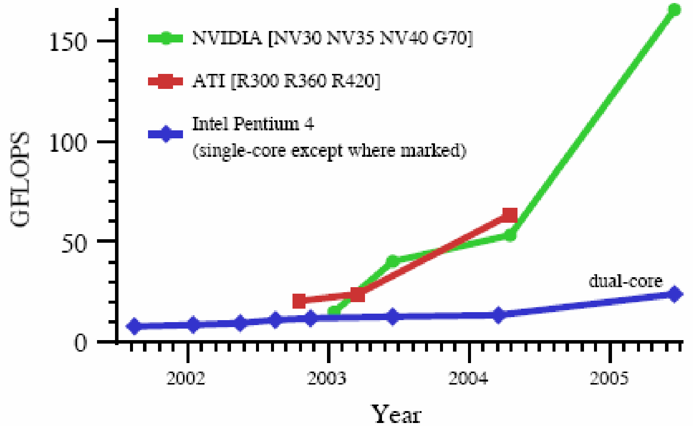
    Intel P4 EE 178M v.s. nVIDIA 7800 302M

Why GPGPU?

- GPU

Why GPGPU?

- Tremendous memory bandwidth and computational power
  - nVIDIA 6800 Ultra: 35.2GB/sec of memory bandwidth
  - ATI X800 XT: 63GFLOPS
  - Intel Pentium4 3.7GHz: 14.8 GFLOPS



Why GPGPU?

- GPU is also accelerating quickly
  - CPU: 1.4x for every year
  - GPU: 1.7x ~ 2.3x for every year
- The disparity in performance between GPU & CPU
  - CPU: optimized for high performance on sequential codes (caches & branch prediction)
  - GPU: higher arithmetic intensity for parallel nature

Why GPGPU?

- Flexible and programmable

  - it fully supports vectorized floating-point operations at IEEE single precision

  - high level languages have emerged

  - additional levels of programmability are emerging with every generation of GPU (about every 18 months)

  - an attractive platform for general-purpose computation

Why GPGU?

- Applications
  - scientific computing
  - signal processing
    - image processing
    - video processing
    - audio processing
  - physically-based simulation
  - visualization
  - …

Why GPGPU?

- Limitations and difficulties

  - the arithmetic power of the GPU is a result of its highly specialized architecture (parallelism)

  - no integer data operands

  - no bit-shift and bitwise operations

  - no double-precision arithmetic

  - an unusual programming model

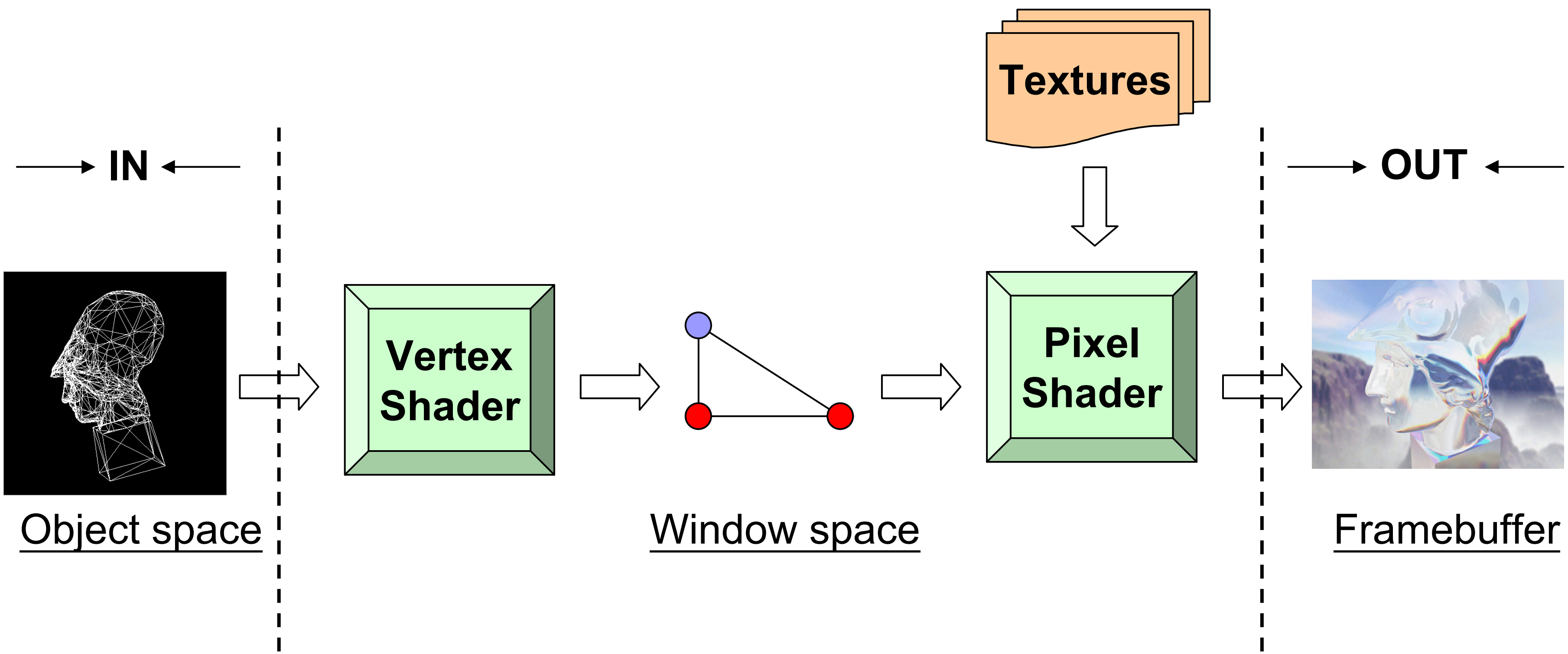
  - these difficulties are intrinsic to the nature of graphics hardware, not simply a result of immature technology

Outline

- Why GPGPU?
- Programmable Graphics Hardware
- Programming Systems
- Writing GPGPU Programs
- Examples
- References

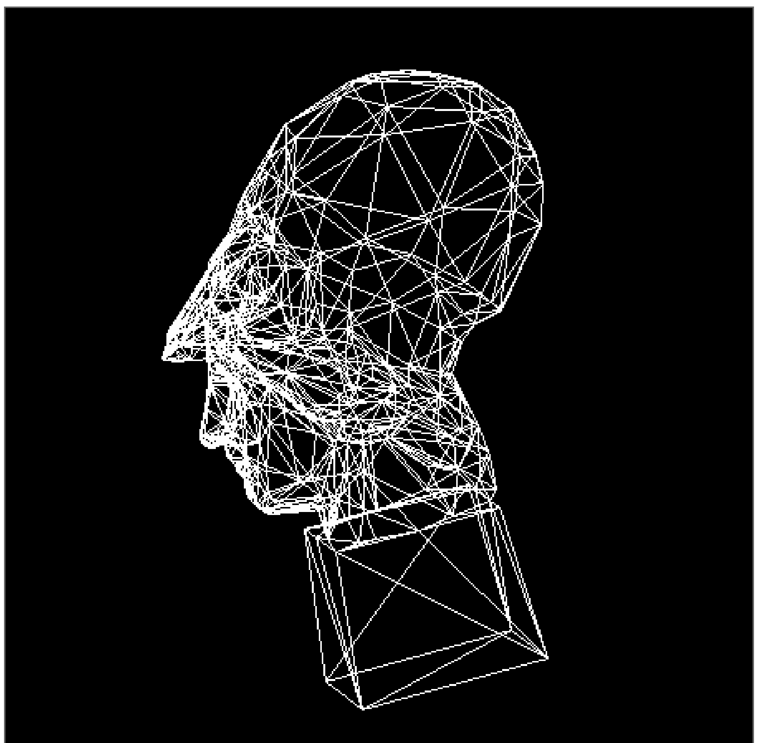
Programmable Graphics Hardware

- Graphics pipeline (simplified)



Programmable Graphics Hardware

- Graphics pipeline



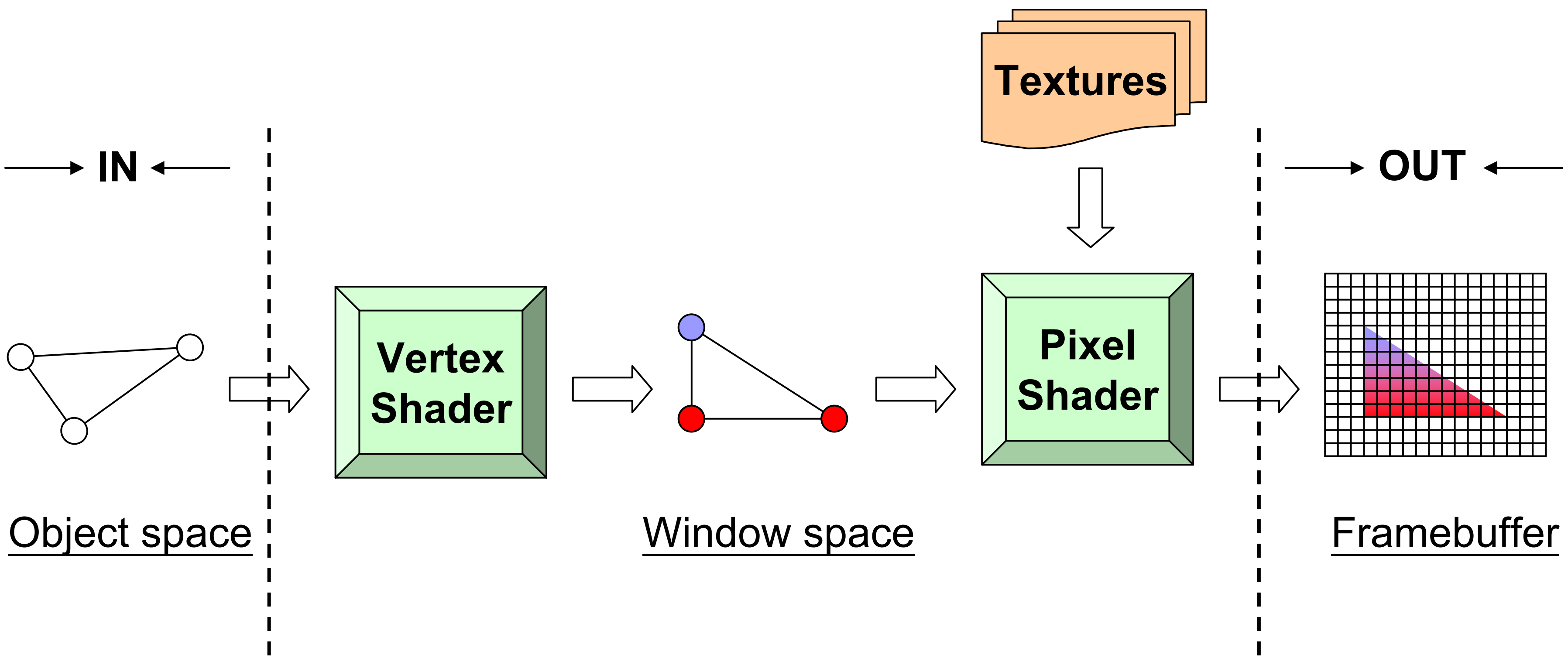
```
v  -1943.297363 -281.849670 435.762909
v  -2081.436035 -281.723267 363.743317
v  -1445.912109 281.329681 644.545166
…

vn  -0.221051 0.258340 -0.940424
vn  -0.220863 0.258493 0.940426
vn  -0.220848 0.030928 -0.974818
…

f 1421//3282 1268//3464 1425//3646
f 1266//4180 1425//3646 1268//3464
f 1266//4180 1264//4343 1425//3646
f 1424//3294 1425//3646 1264//4343
f 1264//4343 1262//4275 1424//3294
…
```

Programmable Graphics Hardware

- Graphics pipeline

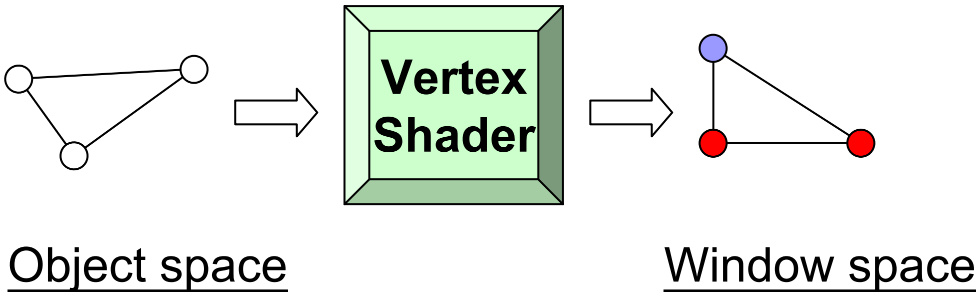
Programmable Graphics Hardware

- Graphics pipeline (simplified)



Programmable Graphics Hardware

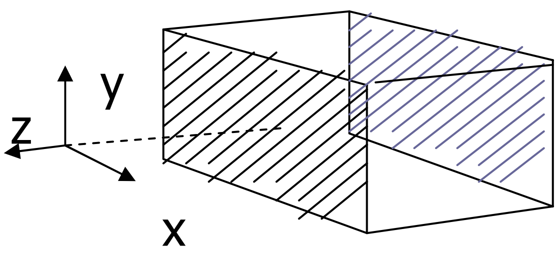
- Vertex shader:
  - modeling transform
  - view transform
  - projection transform



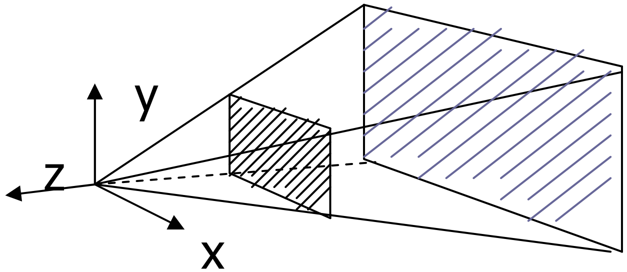
- Projection transform
  - orthogonal projection
  - perspective projection

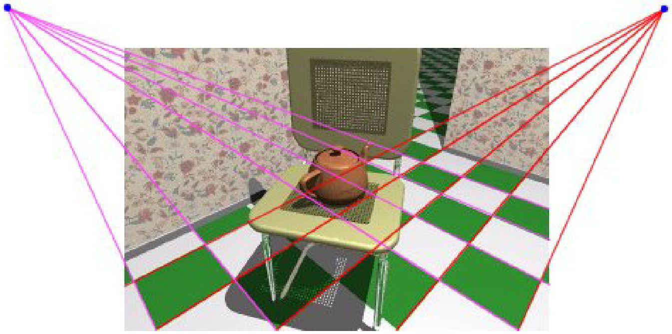
Programmable Graphics Hardware

- Orthogonal projection



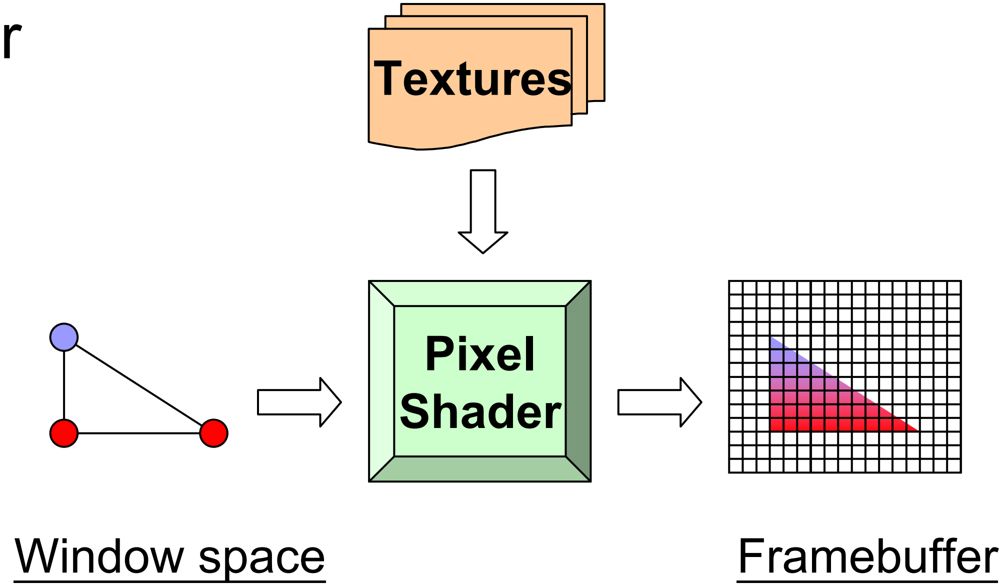
- Perspective projection





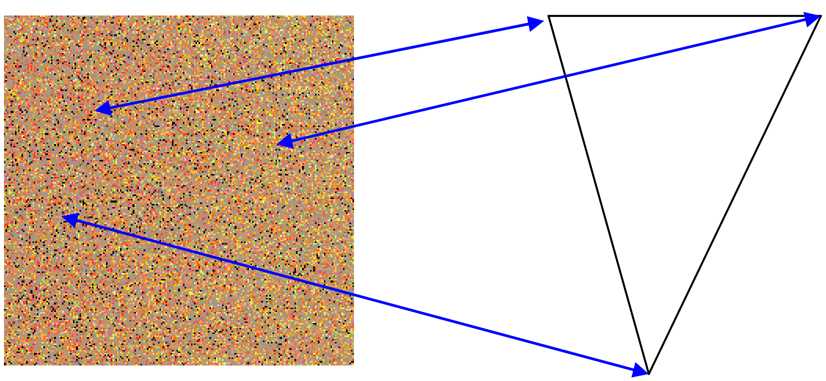
Programmable Graphics Hardware

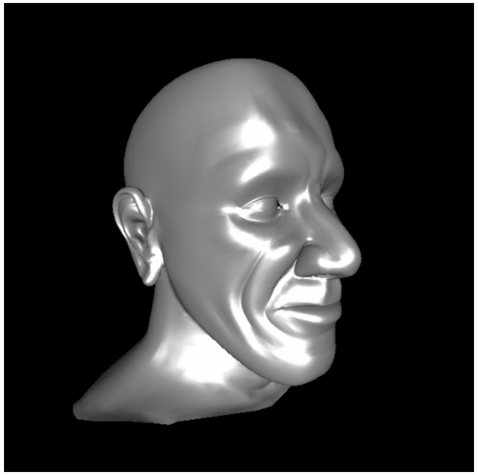
- Pixel shader
  - per pixel operation
  - texture lookup / texture mapping
  - output to framebuffer



Programmable Graphics Hardware

- Texture mapping





Programmable Graphics Hardware

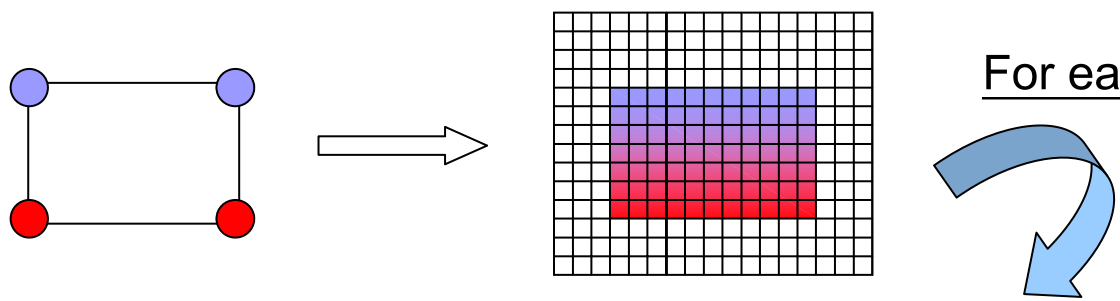
- GPGPU programming model
  - use the pixel shader as the computation engine
  - CPU / GPU analogies:

Data Array	=>	Texture
Memory Read	=>	Texture Lookup
Loop body	=>	Shader Program
Memory Write	=>	Render to framebuffer

  - restricted I/O: arbitrary read, limited write
  - program invocation

Programmable Graphics Hardware

- Program invocation




```
for (int j = 1; j < height - 1; ++j)
{
  for (int i = 1; i < width - 1; ++i)
  {
    // get velocity at this cell
    Vec2f v = grid(x, y);

    // trace backwards along velocity field
    float x = (i - (v.x * timestep / dx));
    float y = (j - (v.y * timestep / dy));

    grid(x,y) = grid.bilerp(x, y);
  }
}
```

C++

```
void advect(float2       uv    : WPOS,
        out float4       xNew  : COLOR,

    uniform float        dt, // timestep
    uniform float        dx, // grid scale
    uniform samplerRECT  u,  // velocity
    uniform samplerRECT  x)  // state
{
  // trace backwards along velocity field
  float2 pos = ub - dt * f2texRECT(u, uv) / dx;

  xNew = f4texRECTbilerp(x, pos);
}
```

Cg

Outline

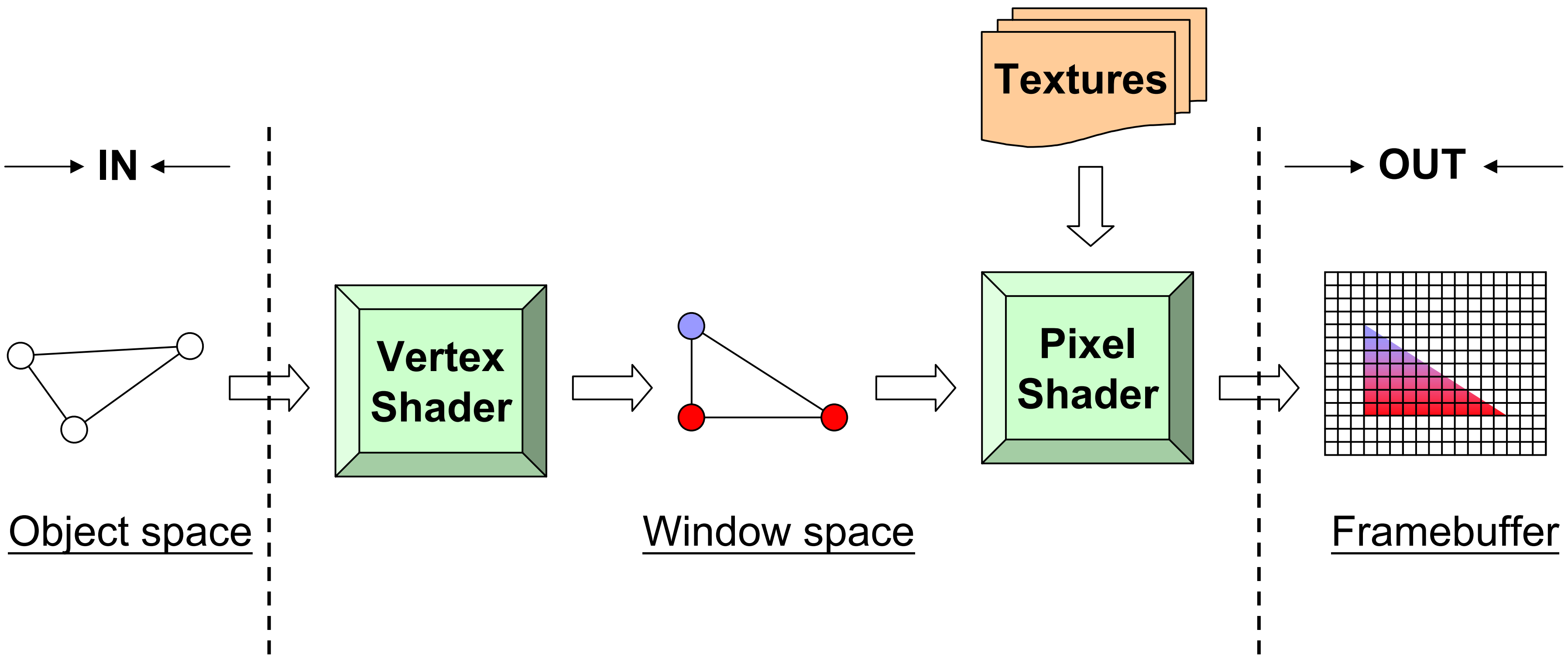
- Why GPGPU?
- Programmable Graphics Hardware
- Programming Systems
- Writing GPGPU Programs
- Examples
- References

Programming Systems

- High-level language
  - write the GPU program
  - nVIDIA Cg / Microsoft HLSL / OpenGL Shading Language
- 3D library
  - build the graphics pipeline
  - OpenGL / Direct3D
- Debugging tool
  - few / none

Programming Systems

- Cg and OpenGL will be used in this tutorial



Outline

- Why GPGPU?
- Programmable Graphics Hardware
- Programming Systems
- Writing GPGPU Programs
- Examples
- References

Writing GPGPU Programs

- OpenGL and Cg will be used as examples
- OpenGL
  - cross platforms
  - growing actively in the extension form
- Cg (C for graphics)
  - cross graphics APIs
  - cross graphics hardware

Writing GPGPU Programs

- System requirements for demo programs
  - Cg compiler: http://developer.nvidia.com/object/cg_toolkit.htm
  - GLUT: http://www.xmission.com/~nate/glut.html
  - GLEW: http://glew.sourceforge.net/
  - platform: Win32
  - IDE: Microsoft Visual C++ .Net 2003
  - GPU: nVIDIA 6600 (or higher) with driver v77.72 (or newer) http://www.nvidia.com/

Writing GPGPU Programs

- Installation

  - Cg: download “Cg Installer” and install it

  - in Visual C++, add new paths for include files and library files in Tools\Options\Projects

  - include files:

    C:\Program Files\NVIDIA Corporation\Cg\include

  - library files:

    C:\Program Files\NVIDIA Corporation\Cg\lib

  - link with cg.lib and cggl.lib

Writing GPGPU Programs

- Installation

  - GLUT: download “glut-3.7.6-bin.zip” and put related files in proper directories

  - header file: C:\$(VCInstallDir)\include\gl
  - library file: C:\$(VCInstallDir)\lib
  - dll file: C:\WINDOWS\system32
  - link with glut32.lib

Writing GPGPU Programs

- Installation

  - GLEW: download binaries and put related files in proper directories

  - header file: C:\$(VCInstallDir)\include\gl
  - library file: C:\$(VCInstallDir)\lib
  - dll file: C:\WINDOWS\system32
  - link with glew32.lib

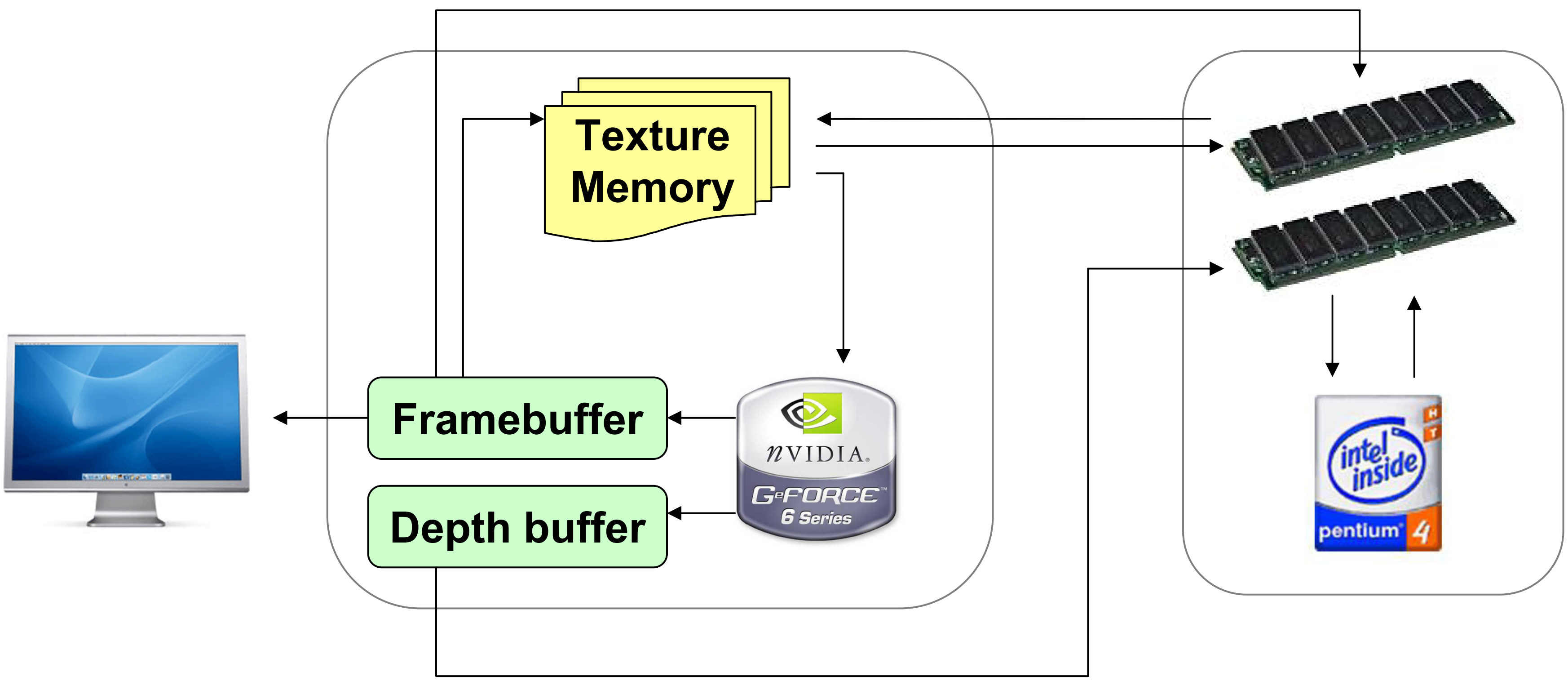
Writing GPGPU Programs

- Syntax highlight in Visual C++ .Net 2003

  - copy the usertype.dat file to
    Microsoft Visual Studio .Net 2003\Common7\IDE
  - open up the registry editor and go to
    HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\VisualStudio\7.1\Languages\File Extensions
  - copy the default value from the .cpp key
  - create a new key under the File Extensions with the name of .cg
  - paste the value you just copied info the default value

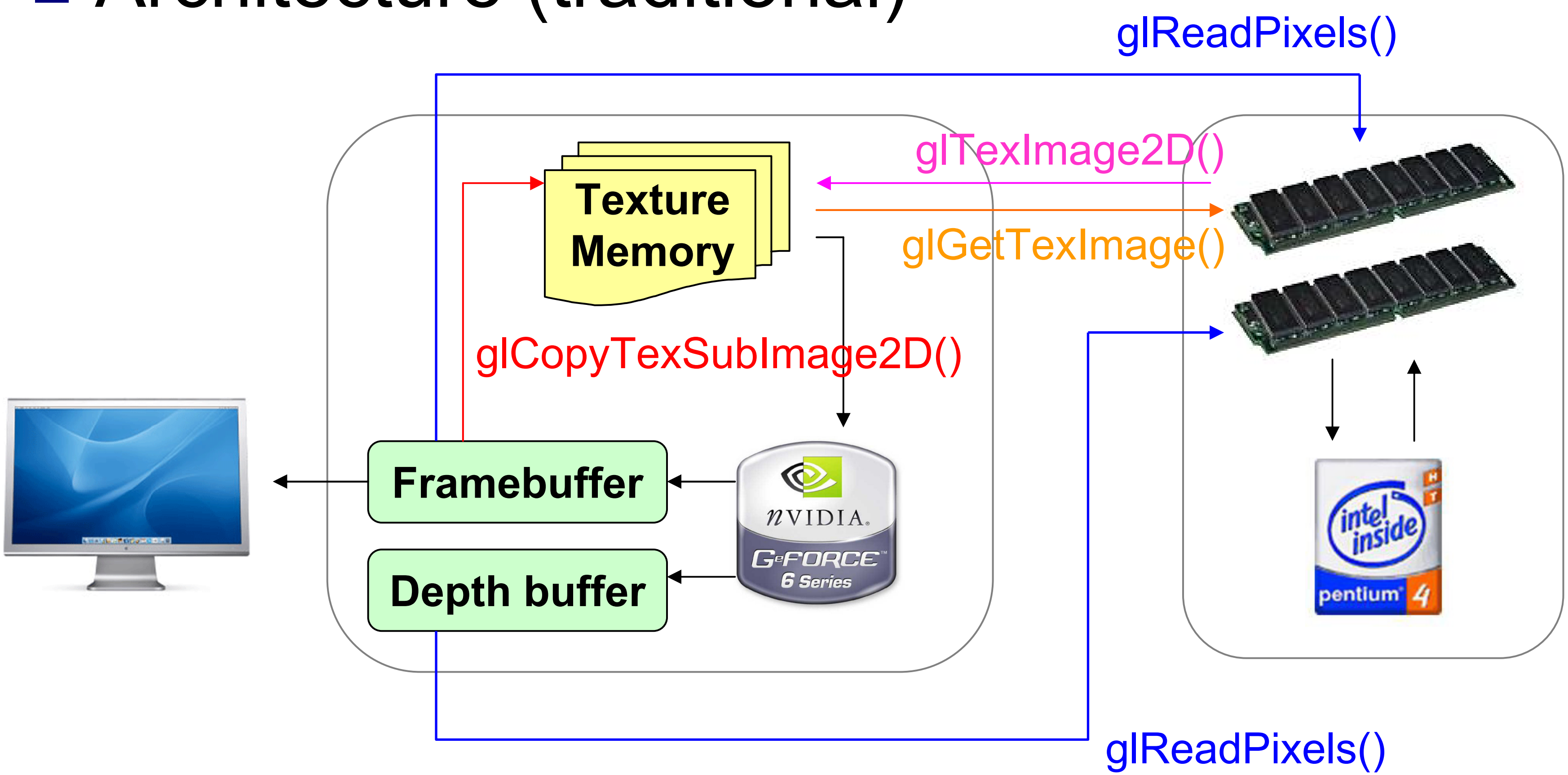
Writing GPGPU Programs

- Architecture (traditional)



Writing GPGPU Programs

- Architecture (traditional)

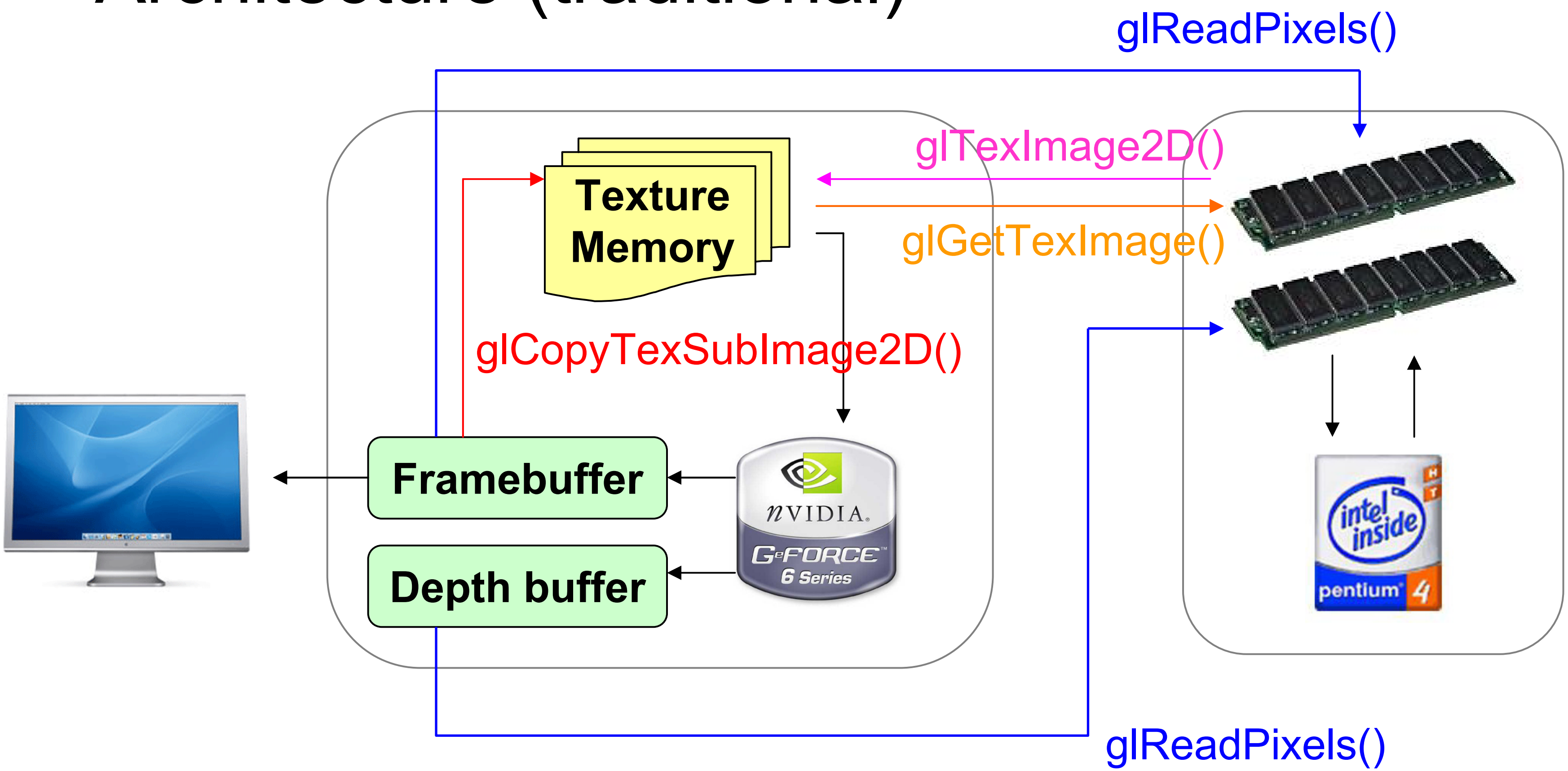


Writing GPGPU Programs

- Uploading is fast
  - uploading: glTexImage2D()
- Downloading is extremely slow
  - downloading: glReadPixels(), glGetTexImage()
- GPU can only render to framebuffer and depth buffer
  - if one wants to store the output in a texture, glCopyTexSubImage2D() must be called

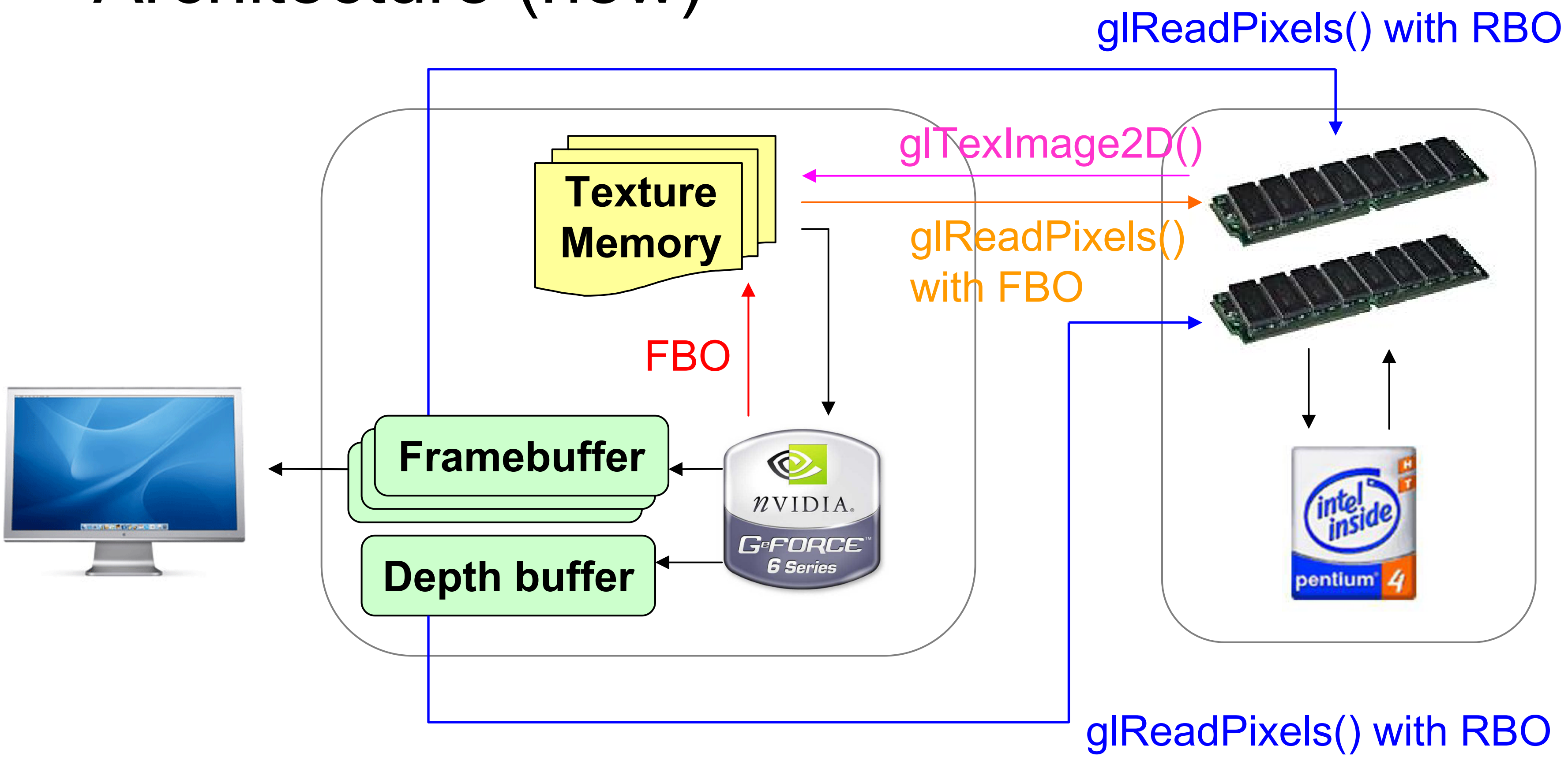
Writing GPGPU Programs

- Architecture (traditional)



Writing GPGPU Programs

- Architecture (new)



Writing GPGPU Programs

- Uploading is fast (glTexImage2D)
- Downloading is getting fast

  - with FBO / RBO extensions, glReadPixels() is speeding up (forget about PBO – Pixel Buffer Object)
- GPU is able to render not only to framebuffer and depth buffer, but also to textures

  - with FBO and MRT extensions

  - forget about pBuffer and RenderTexture

Writing GPGPU Programs

- OpenGL extensions used:

- rectangle texture	(NPOT texture)
- floating-point texture	(prevent [0, 1] clamping)
- multi-texture	(multiple textures)
- framebuffer object	(FBO, for rendering to texture)
- renderbuffer object	(RBO, for fast downloading)
- multiple render targets	(MRT, for multiple outputs)

Outline

- Why GPGPU?
- Programmable Graphics Hardware
- Programming Systems
- Writing GPGPU Programs
- Examples
- References

Examples

- 6 examples

  OpenGL:

  - 1. texture mapping
  - 2. texture mapping with FBO and RBO

  OpenGL and Cg:

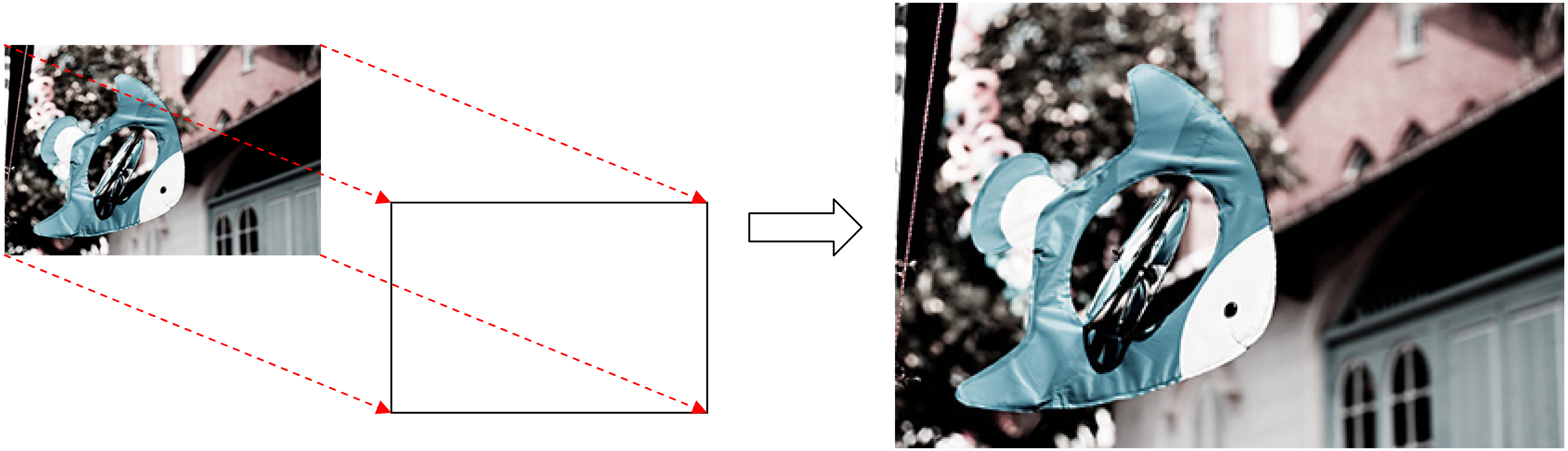
  - 3. image warping
  - 4. image blurring
  - 5. image blending
  - 6. MRT

Example #1

- Texture mapping
  - OpenGL introduction
  - GLUT and WGL
  - rectangle texture
  - image I/O for GPU

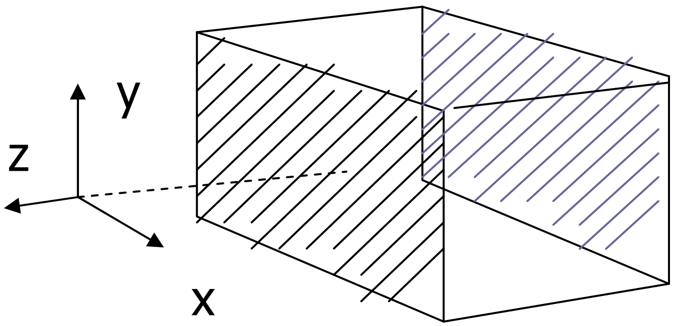
Example #1

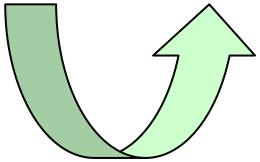
- Texture mapping



Example #1

- Viewport transformation





Example #1

- Texture creation

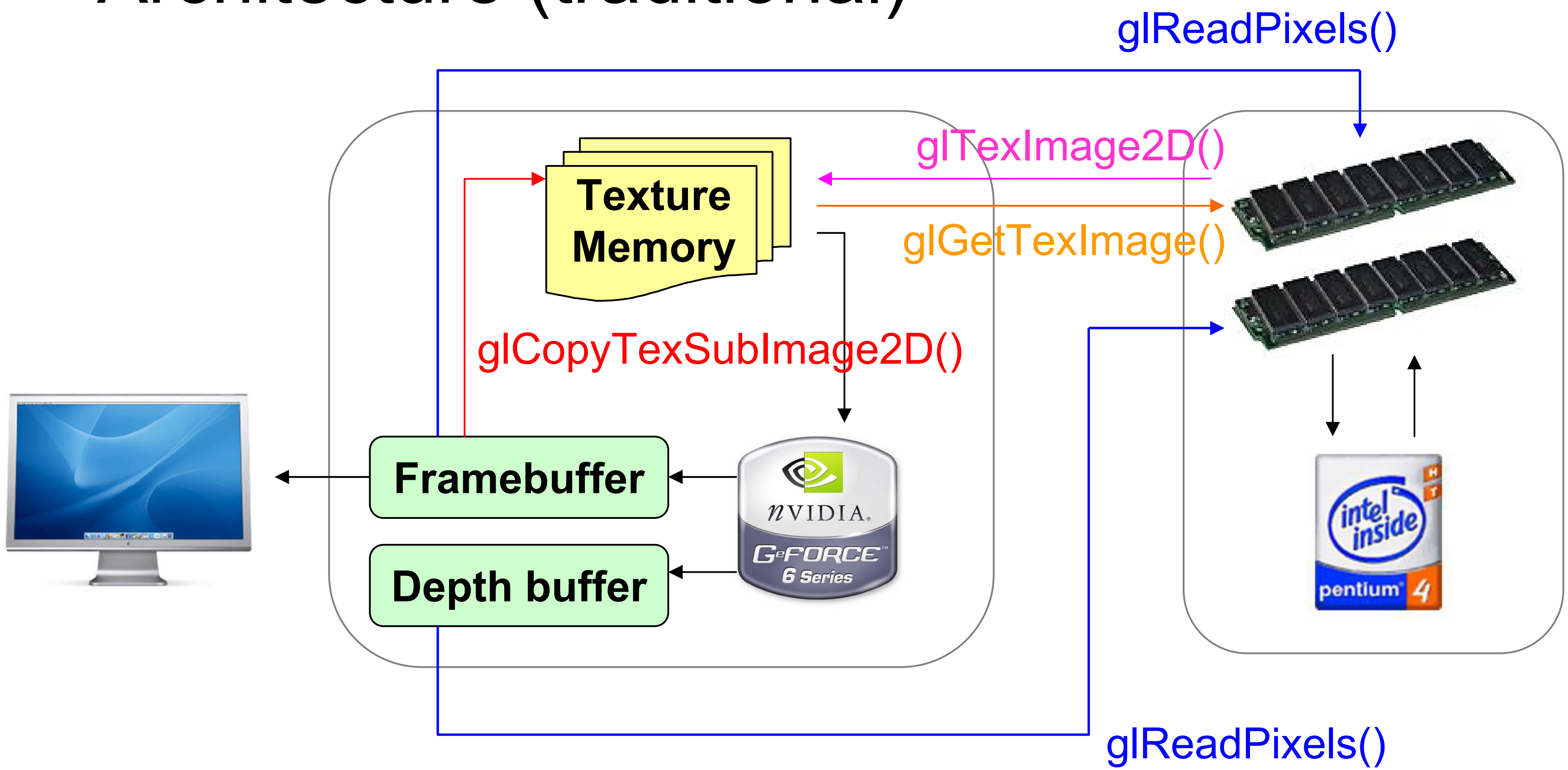
  - generate a texture

  - setup the texture properties

  - upload an image from the main memory to the GPU

Example #1

- Architecture (traditional)

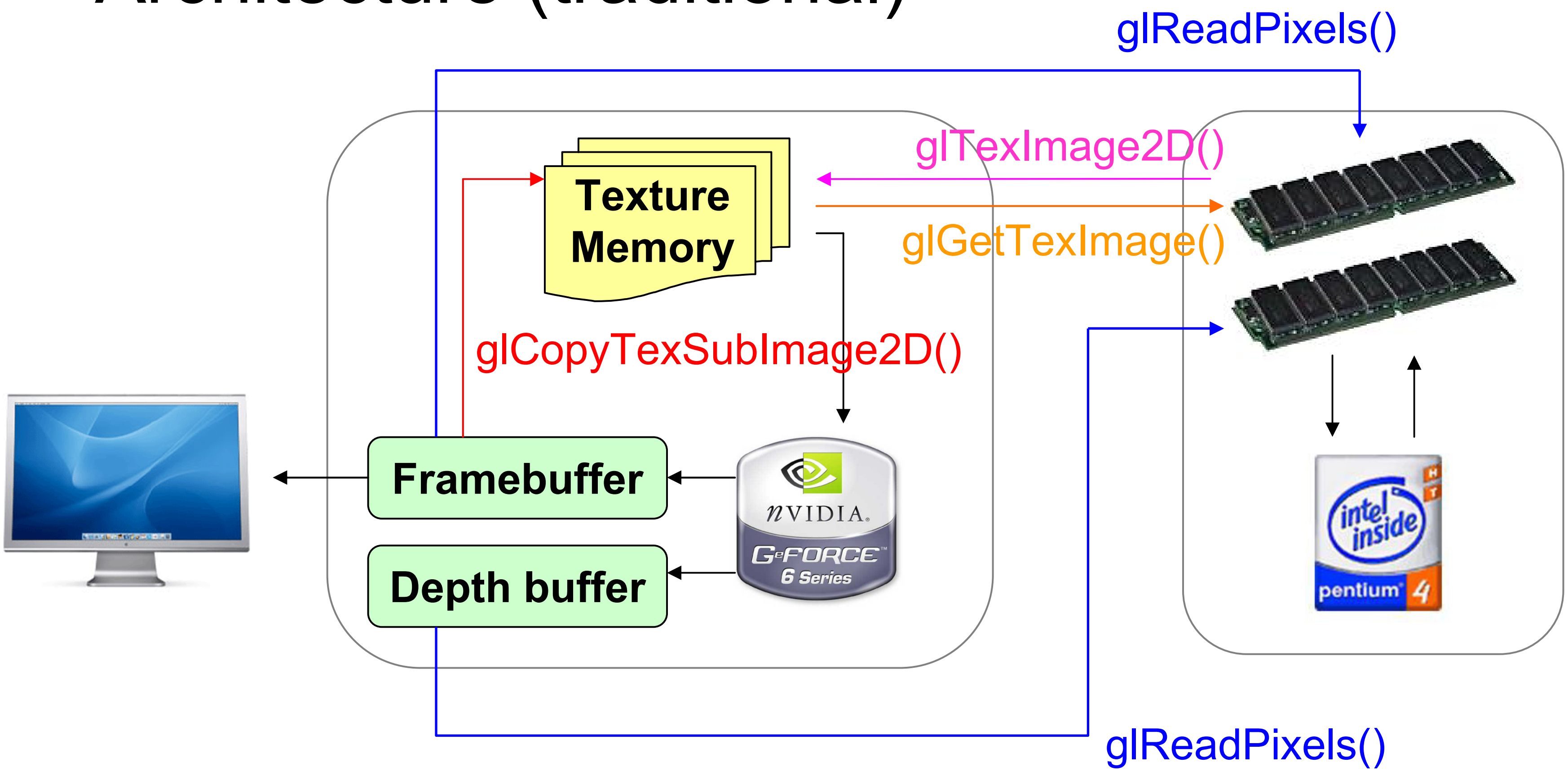


Example #2

- Texture mapping with FBO and RBO
  - render to texture with FBO
  - fast downloading with RBO

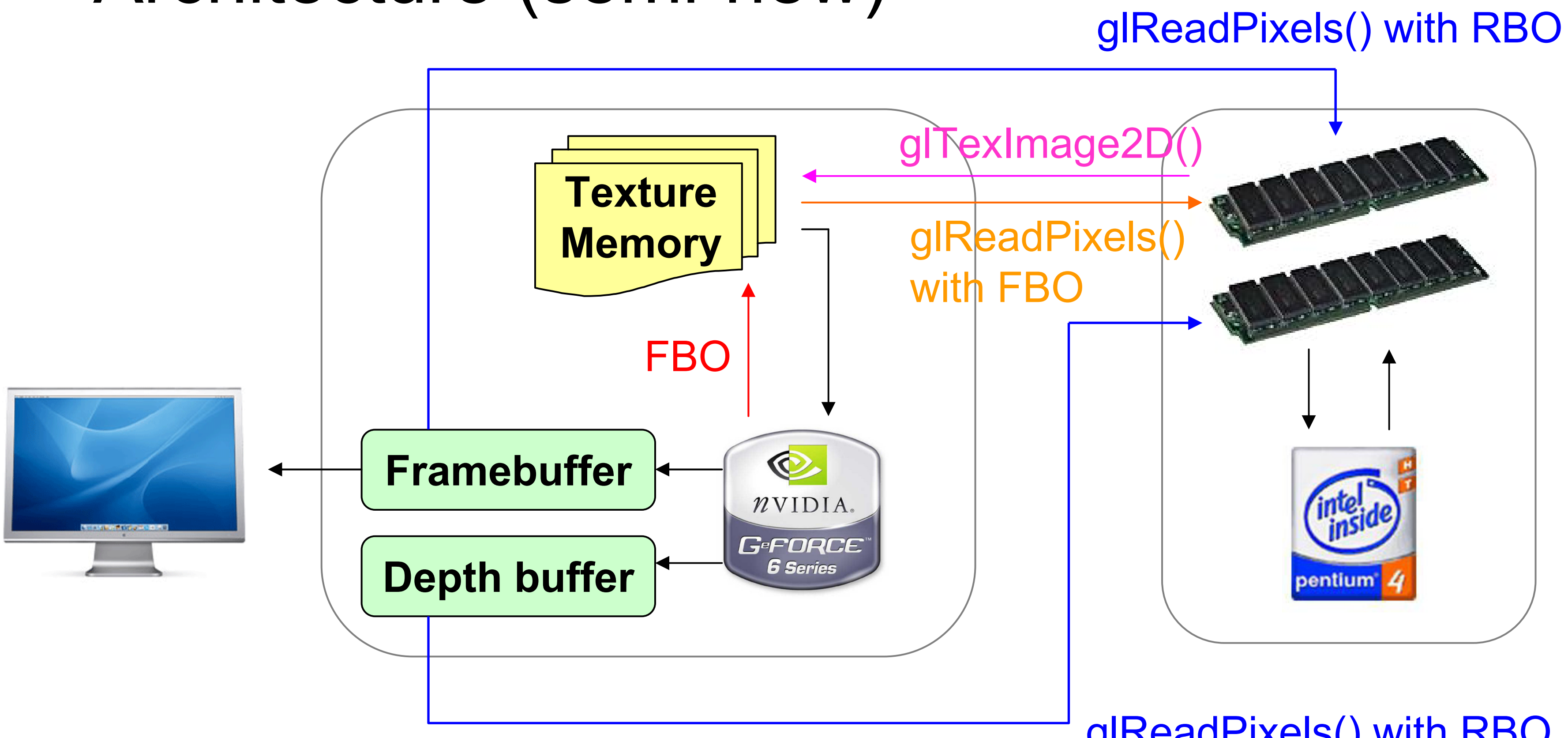
Example #2

- Architecture (traditional)



Example #2

- Architecture (semi-new)



Example #2

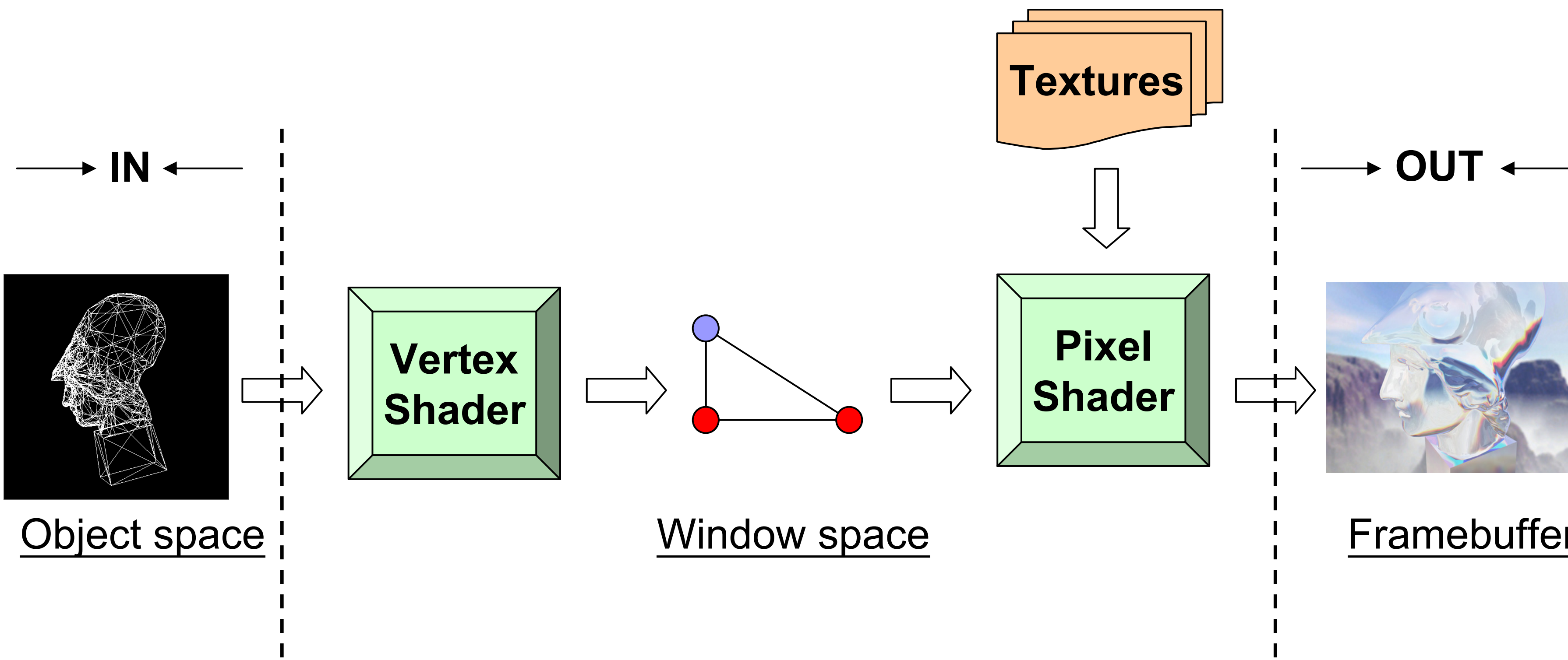
- FBO creation
  - generate an FBO
  - generate a texture
  - associate the texture with the FBO
- RBO creation
  - generate an RBO
  - allocate memory for the RBO
  - associate the RBO with the FBO

Example #3 and #4

- Image warping and image blurring
  - Cg introduction
  - environment setup
  - Cg runtime
  - Cg standard library

Example #3 and #4

- Graphics pipeline (simplified)



Example #3 and #4

- Cg runtime

  - environment setting, program compiling/loading, and parameters passing

- Cg standard library

  - mathematical functions

  - geometric functions

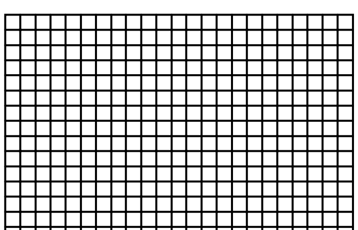
  - texture map funcitons

Example #3

- Forward warping
  - straight forward
  - holes in the destination image

x **M** ~=

- Backward warping
  - make sure that there would be no holes in the destination image
  - interpolation is needed



x $\mathbf{M}^{-1}$ to lookup

=

Example #4

- Image blurring

  - box filter

  - the value of a destination pixel is the weighted average of its neighboring pixels in the source image

Example #3 and #4

- Cg language

  - vector data type (SIMD)

    > e.g. `float4 var;`

    then we have `var.xyzw` or `var.rgba`

    > e.g. `float2 position = 3 * var.xz;`

  - semantics: `TEX0`, `COLOR`…

  - type qualifier: `out`, `uniform`…

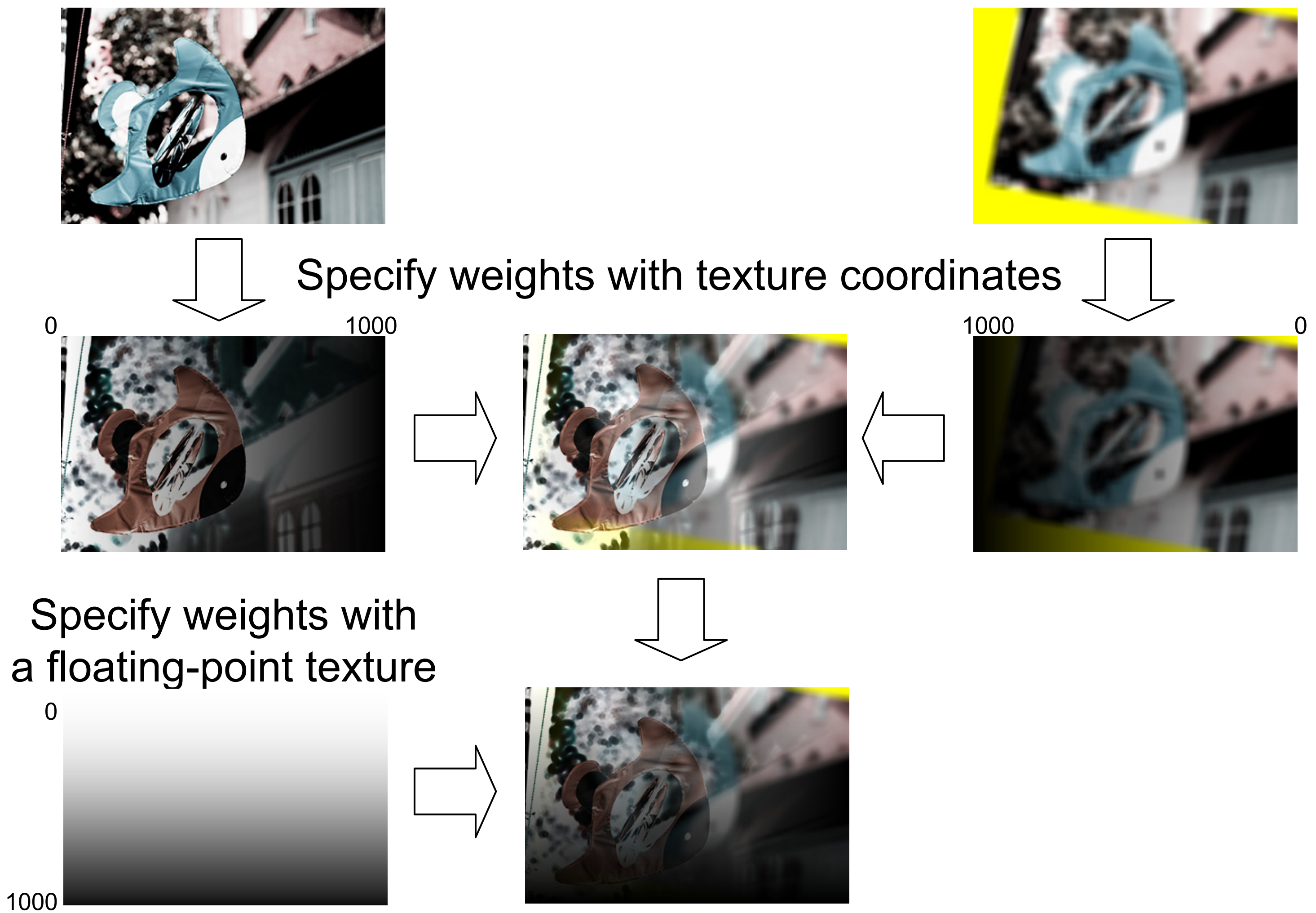
Example #5

- Image blending
  - floating-point texture
  - multi-texture

Example #5

- Floating-point texture
  - get more precision (16-bit or 32-bit) than only 8-bit
  - especially useful in GPGPU
- Multi-texture
  - inherent in Cg for multi-texture accessing
  - what counts is the multi-texture “coordinates”
  - send more information to the GPU
  - linear-interpolated data

Example #5


Specify weights with texture coordinates
0
1000
1000
0
Specify weights with
a floating-point texture
0
1000

Example #5

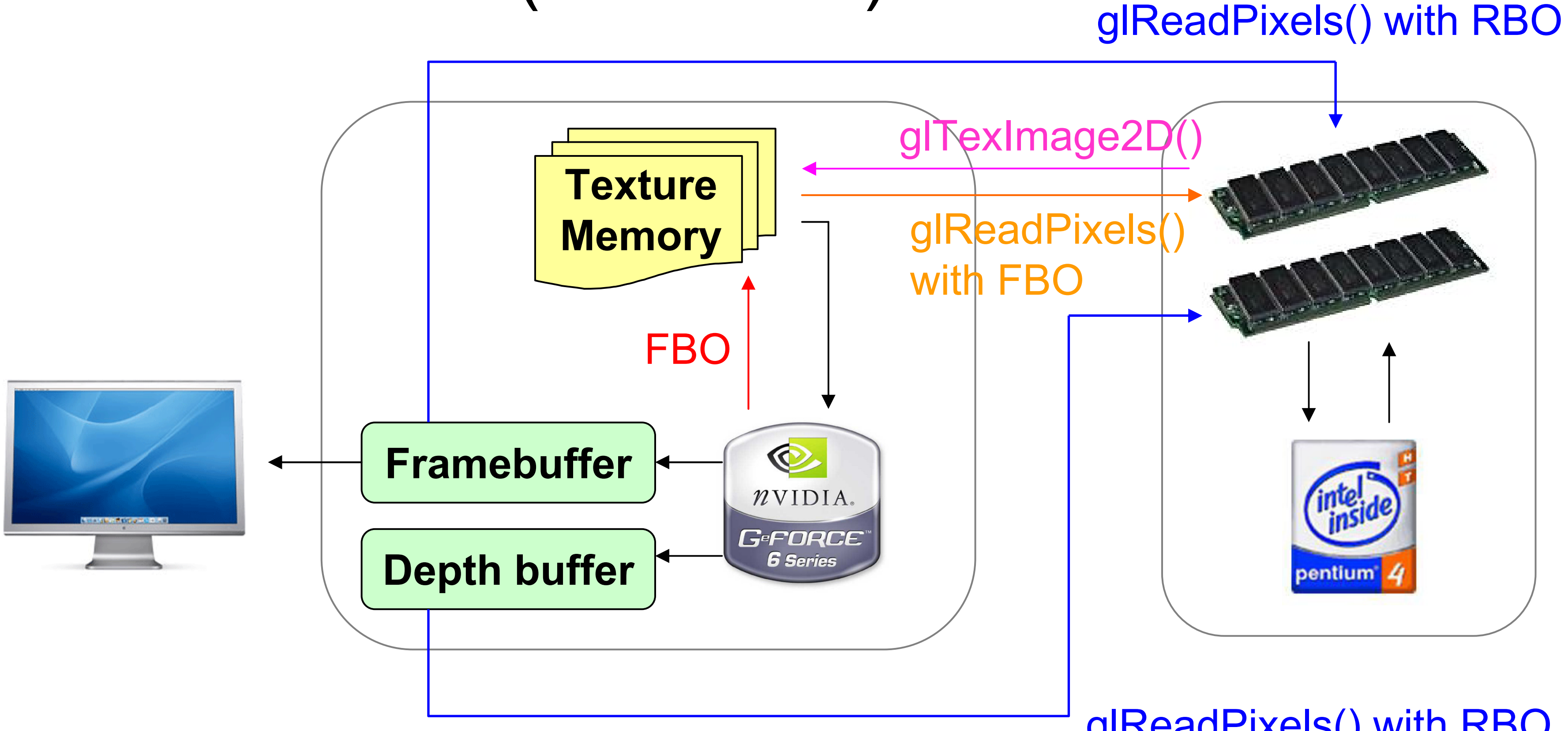
- Depth buffer readback

  - not really useful since another FBO/RBO is needed

- Floating-point texture readback
  - glReadPixels() must be *inside* the FBO
  - use GL_NEAREST for a floating-point texture

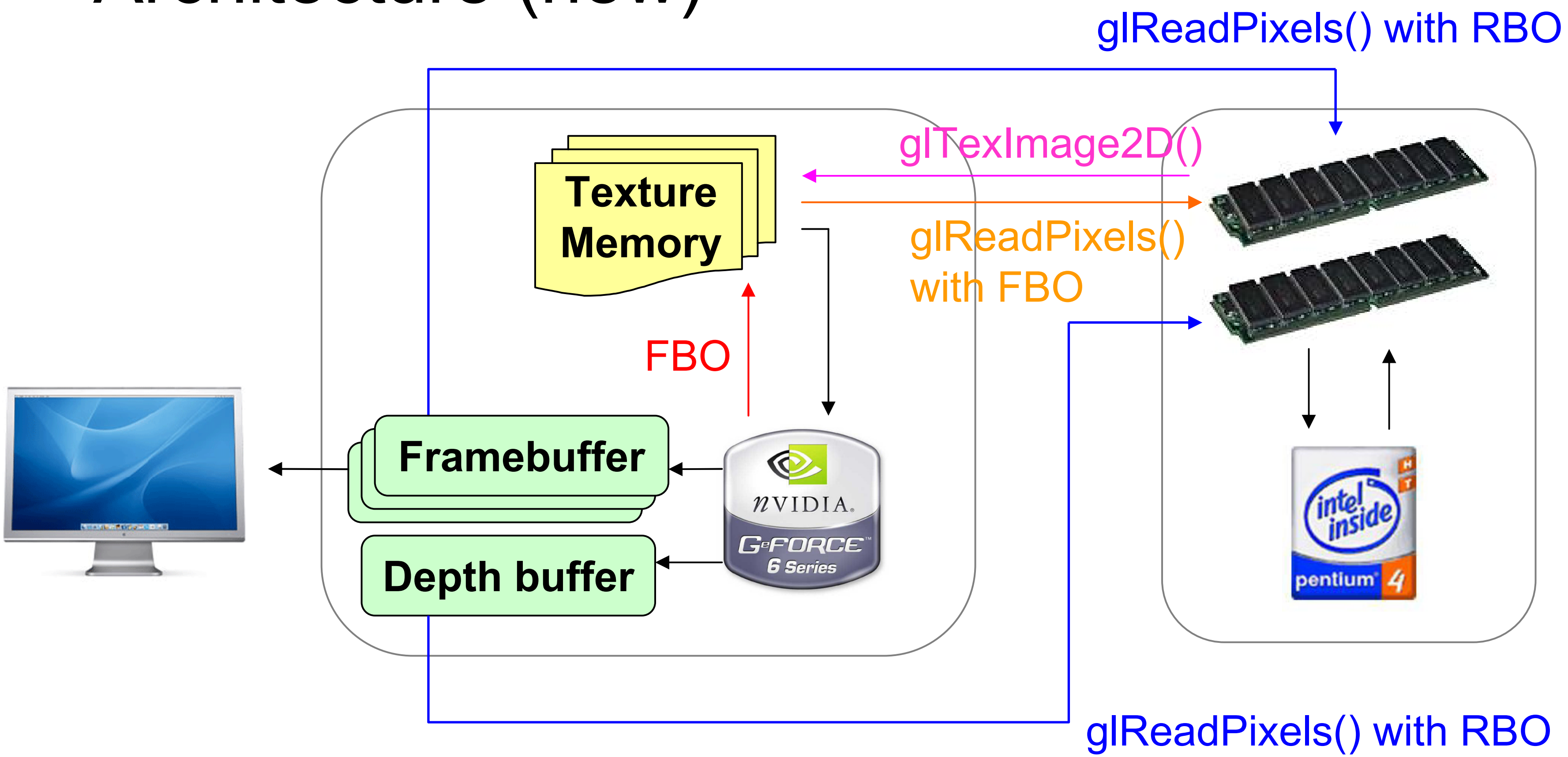
Example #5

- Architecture (semi-new)



Example #5

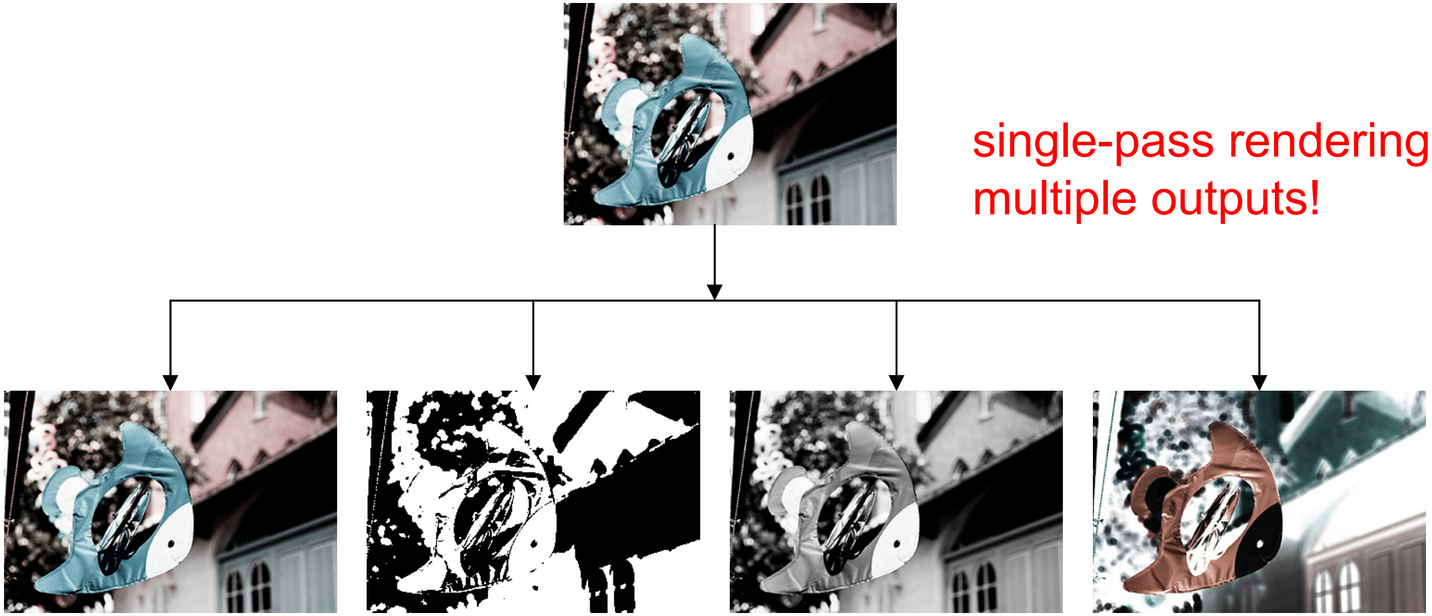
- Architecture (new)



Example #6

- MRT

  \- multiple render targets



single-pass rendering,
multiple outputs!

Example #6

- The format of the render targets must be the same
- Associate different color attachments with the FBO
- MRT operation

  - use glDrawBuffers() to activate the MRT

  - use glReadBuffer() to specify the buffer for readback

Example #6

- Pixel format review

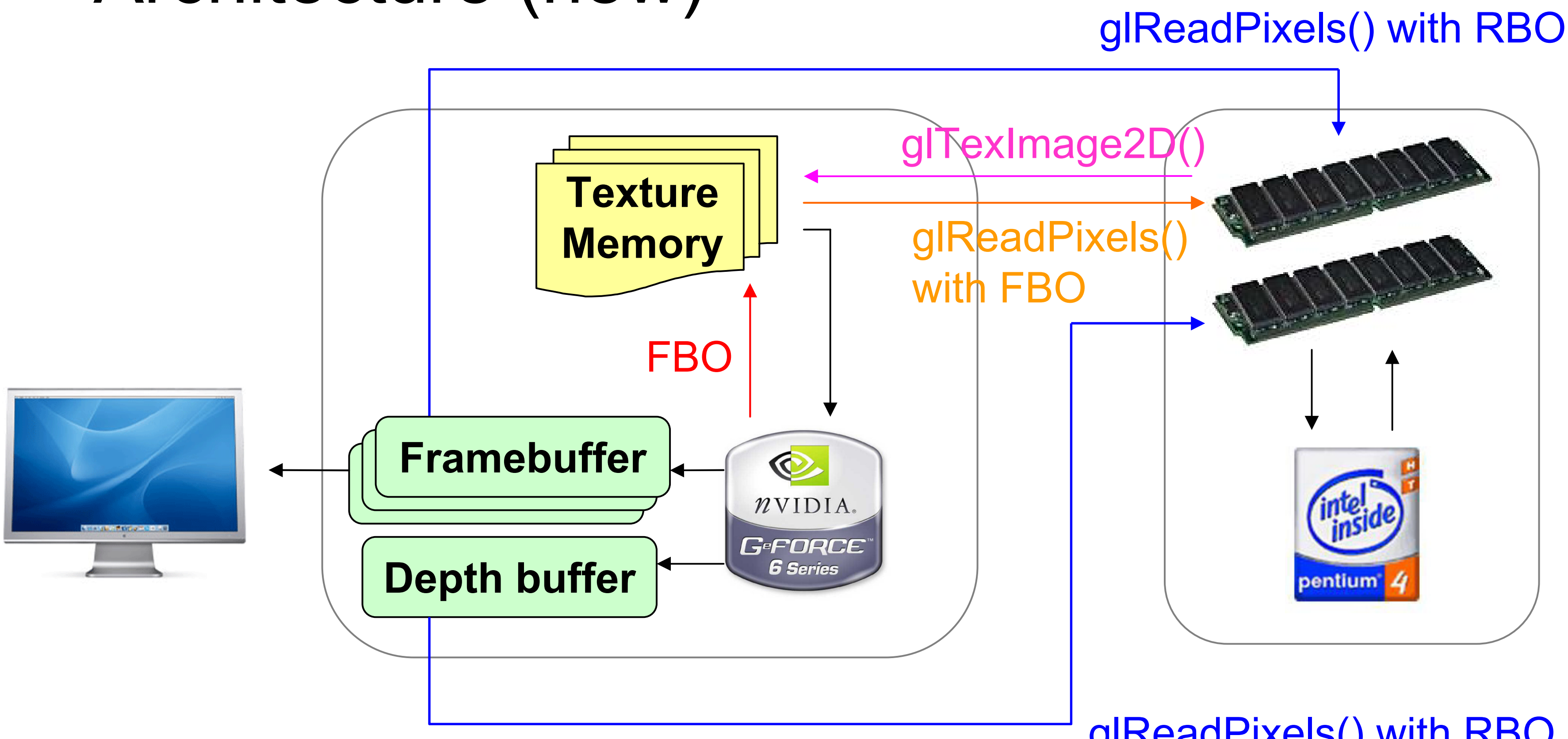
  - clamp-free and truly floating-point range are available while GL_RGBA32F_ARB or GL_RGBA16F_ABR with GL_FLOAT uploading and/or downloading are used

  - uploading with GL_UNSIGNED_BYTE will cause [0, 255] => [0, 1] no matter what the internal format is

  - without the floating-point texture, what read back with GL_FLOAT would be clamped to [0, 1]

Example #6

- Architecture (new)



Examples

- Tips for GPU programming

  - balance the loading between CPU and GPU

  - use branch judiciously

  - data type with lower precision

  - reduce the I/O between CPU and GPU, especially for downloading

  - SIMD operation

  - do not forget the standard library

  - linear-interpolation property

Examples

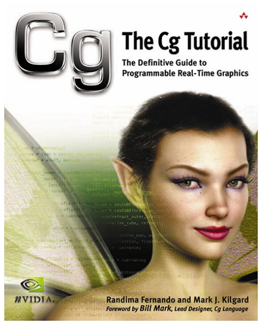
- Conclusion for the procedure of GPGPU programming
  1. wrap data as textures
  2. draw a quadrangle
  3. invocate fragment programs
  4. store GPU outputs as a texture for multi-pass calculation (then go back to step 2)
  5. output the final result to framebuffer or read it back to main memory

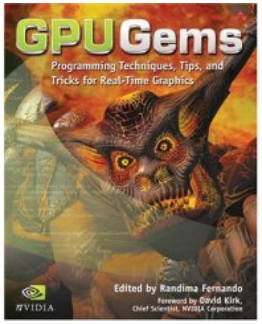
Outline

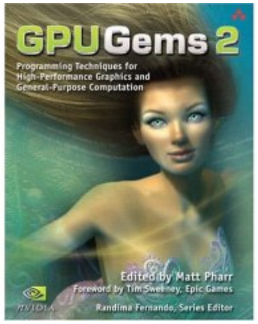
- Why GPGPU?
- Programmable Graphics Hardware
- Programming Systems
- Writing GPGPU Programs
- Examples
- References

References

- Paper
  - *A Survey of General-Purpose Computation on Graphics Hardware*, EUROGRAPHICS 2005
- Website
  - nVIDIA: http://developer.nvidia.com (nVIDIA SDK)
  - GPGPU: http://www.gpgpu.org
- Book
  - *The Cg Tutorial*
  - *GPU Gems 1 & 2*

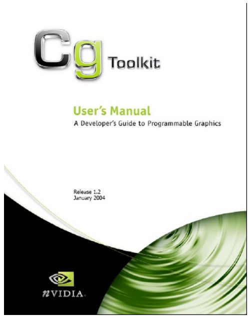






References

- Documentation
  - Cg User Manual
  - *NVIDIA GPU Programming Guide*



- Human Resource (Graphics Group)
  - Wan-Chun Ma, [firebird@cmlab]
  - Cheng-Han Tu, [toshock@cmlab]
  - Pei-Lun Lee, [ypcat@cmlab]