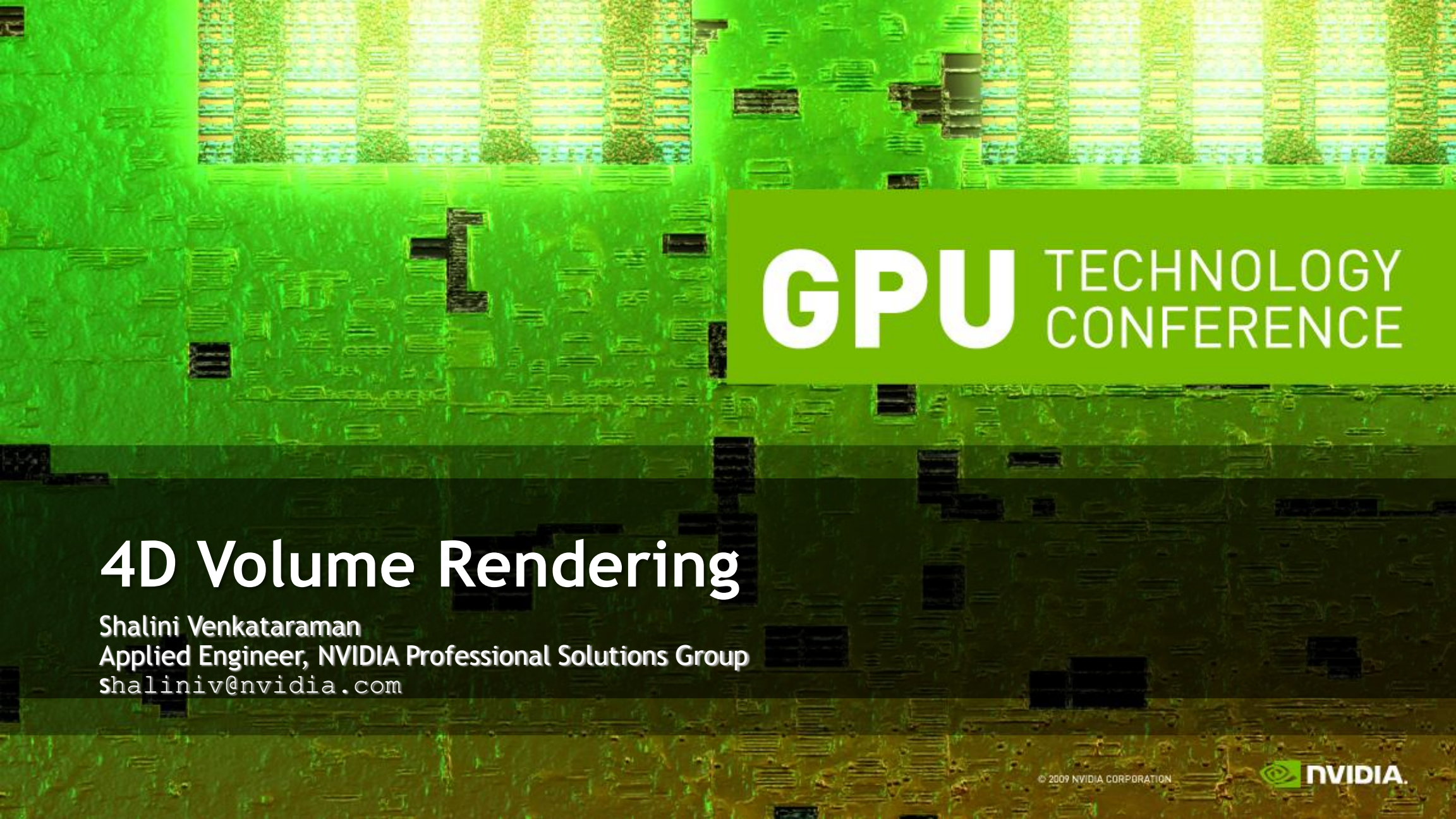


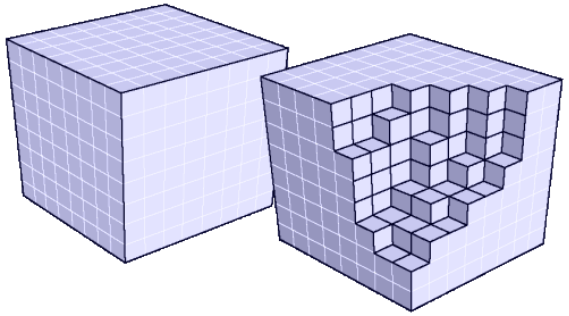
4D Volume Rendering

Shalini Venkataraman
Applied Engineer, NVIDIA Professional Solutions Group
Shaliniv@nvidia.com

NVIDIA.

Motivation

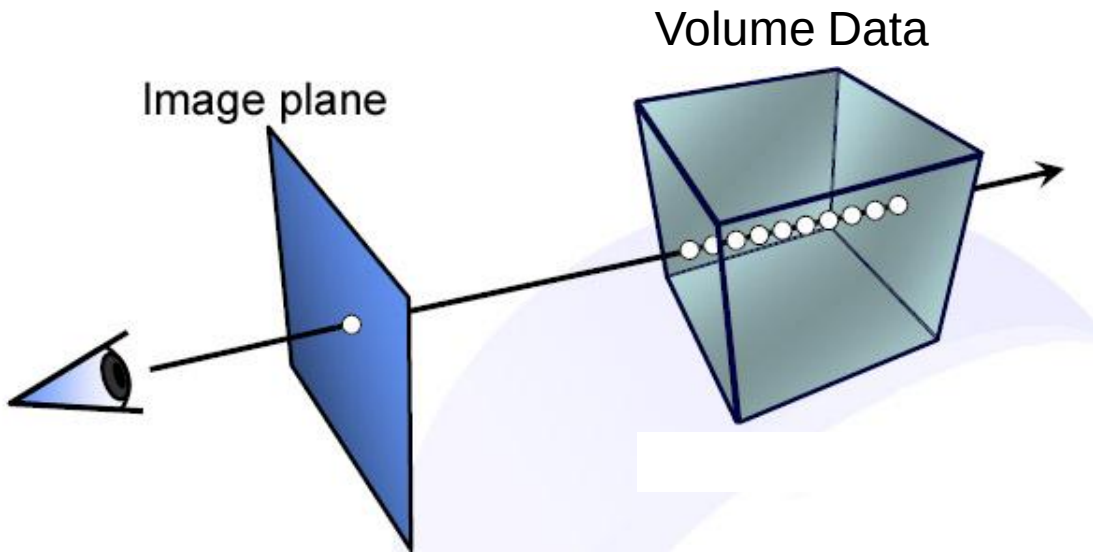
- Large 4D data from
  - Numerical simulations
  - Acquisition devices –scanners etc
- Powerful graphics
  - Increasing graphics capabilities per GPU
    - 5800 has 4GB graphics memory, capable of 256^4 8-bit volumes!
  - Ease in aggregating Multi-GPUs
    - Hardware - Quadroplex SVS
    - Software API's - WGL_Affinity



Outline

- Volume rendering techniques
  - Object-order vs image-order
- Extending to 4D – Single-GPU
- Multi-GPU approaches
  - Spatial/time partitioning
- Stereo

Volume Rendering Approaches



Object-Order Texture Slicing

CPU - Plane-box intersection

GPU – 3d texture & transfer function lookup

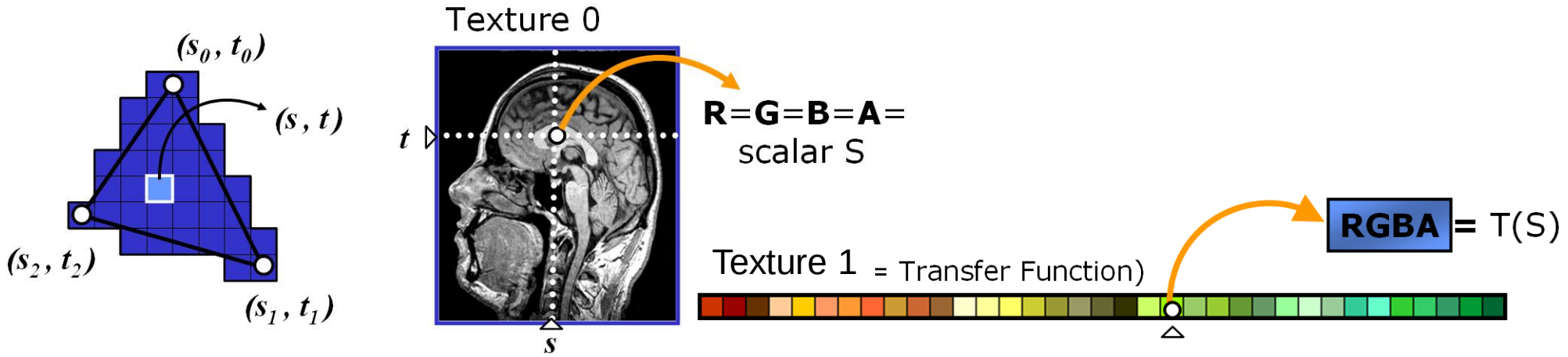
Image-Order Raycasting

CPU – Quad to invoke fragment shader

GPU – All the heavy liftting!

Raycasting + texture lookup

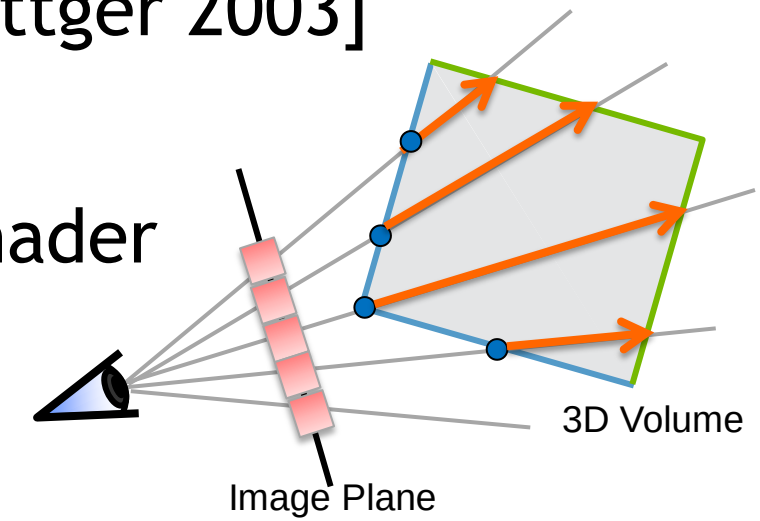
Transfer function lookup




```
//simple shader that looks up rgba from colortable for grayscale volume
uniform sampler3D     volumeTex;         // volume data
uniform sampler1D    transferFuncTex; //color map texture

void main(void)
{
     float intensity = texture3D(volumeTex, gl_TexCoord[0].xyz).a;
     gl_FragColor = texture1D(transferFuncTex, intensity);
}
```

Raycasting - Image Order

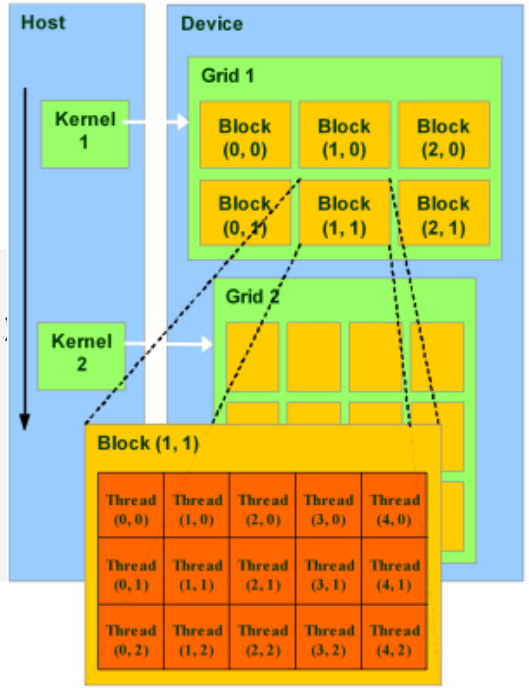
- Single-pass raycasting
  - Compute ray intersection with volume bounding box, advance along camera direction and composite
  - Ray-boundbox calculation [Roettger 2003]
  - Pros - Simple
  - Cons - Full load on fragment shader



CUDA Raycasting

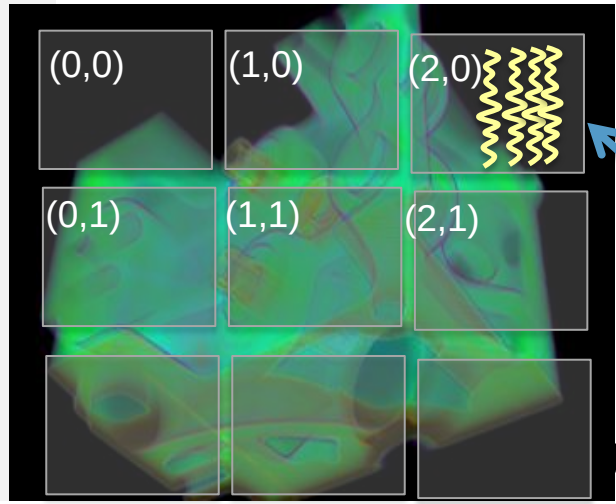
Init

```
dim3 m_blockSize(8,16); // Initialize block size
m_gridSize = dim3(iDivUp(width, m_blockSize.x), iDivUp(height, m_blockSize.y));
// SET VOLUMESIZE AND CHANNEL DESC
cudaMalloc3DArray(&m_dVolumeArray, &channelDesc, volumeSize);
// Copy data from host to cuda array in device (m_dVolumeArray)
// SET COPYPARAMS FOR CUDAMEMCPY
cudaMemcpy3D(&copyParams);
```



Draw

```
//Mapping buffer object (m_pbo) so that kernels can read /write
to buffer
//via the device memory address(d_outputImage)
cudaGLMapBufferObject((void**)&d_outputImage, m_pbo);
cudaMemset(d_outputImage, 0, width*height*4);
// call CUDA kernel, writing results to PBO
d_render<<<m_gridSize, m_blockSize>>>(d_outputImage, width,
height, near, far);
//done with kernel, unmap buffer object
cudaGLUnmapBufferObject(m_pbo);

//BLIT to screen
```

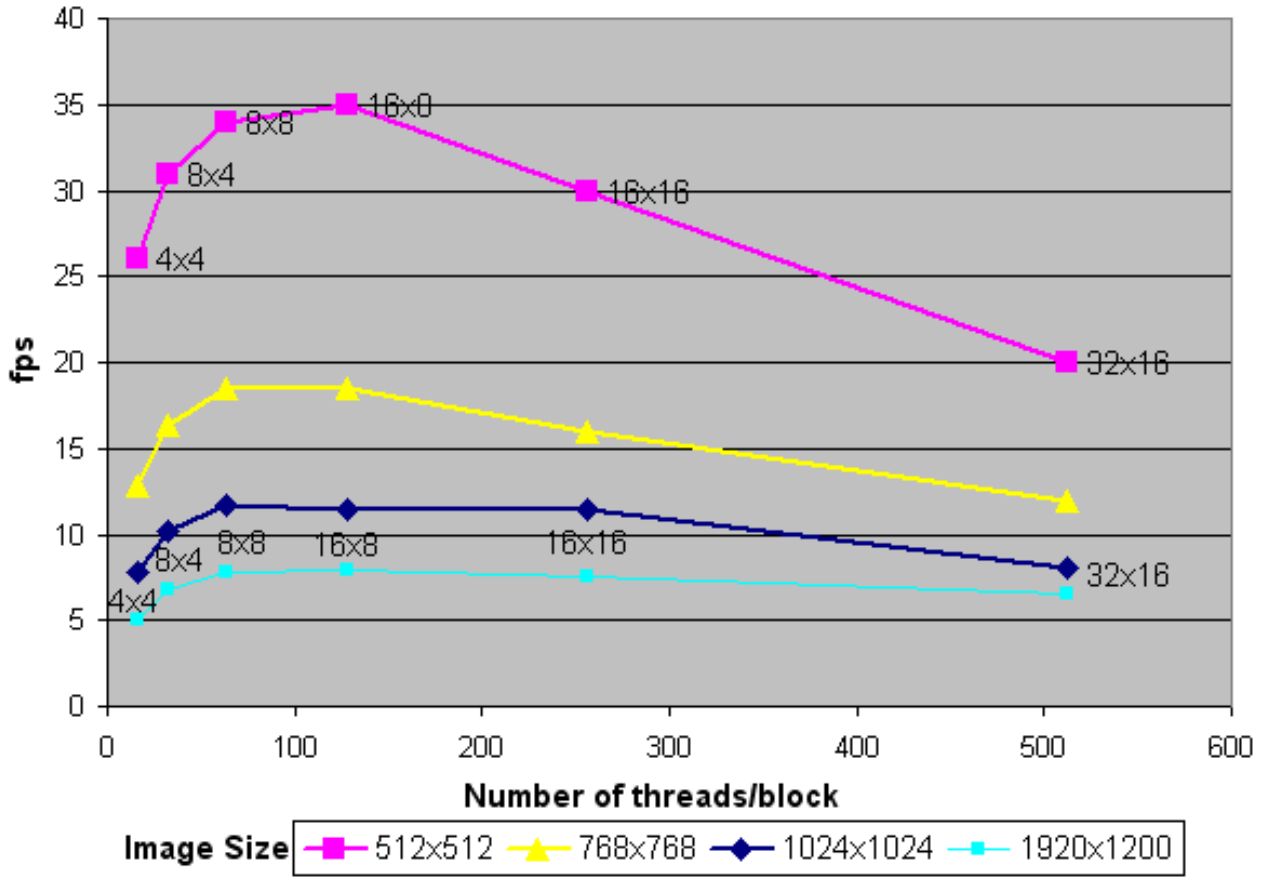


Implementation (in CUDA)

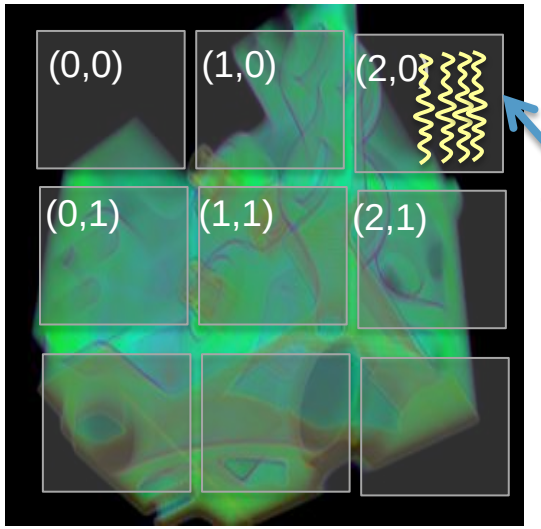
```
__global__ void
d_render(uint *d_outputImage, uint imageW, uint imageH, float frustum_near, float frustum_far)
{
      float3 boxMin = make_float3(0.0f, 0.0f, 0.0f);
      float3 boxMax = make_float3(1.0f, 1.0f, 1.0f);
      uint x = blockIdx.x*blockDim.x + threadIdx.x;
      uint y = blockIdx.y*blockDim.y + threadIdx.y;
      // calculate eye ray in world space
      Ray eyeRay;
      eyeRay.start = c_rayStart;
      eyeRay.dirn = normalize(make_float3(u, v, -frustum_near));
      eyeRay.dirn = mul(c_invViewMatrix, eyeRay.dirn);
      int hit = intersectBox(eyeRay, boxMin, boxMax, &tnear, &tfar); // find intersection with box
      if (!hit) return;
      if (tnear < 0.0f) tnear = 0.0f;     // clamp to near plane
      // march along ray from back to front, accumulating color
      float4 sum = make_float4(0.0f);;
      float t = tfar;
      for(int i=0; i<maxSteps; i++) {
            float3 pos = eyeRay.start + eyeRay.dirn*t;
            float sample = tex3D(volumeTex, pos.x, pos.y, pos.z); // read from 3D texture
            float4 col = tex1D(transferTex, sample); // lookup in transfer function texture
            sum.xyz= lerp(sum.xyz, col.xyz, col.w); // accumulate color
            sum.w = lerp(sum.w, 1.0, col.w); // accumulate opacity
            t -= tstep;
            if (t < tnear) break;
      }
      d_outputImage[y* imageW+ x] = rgbaFloatToInt(sum);
}
```

From NVIDIA CUDA SDK

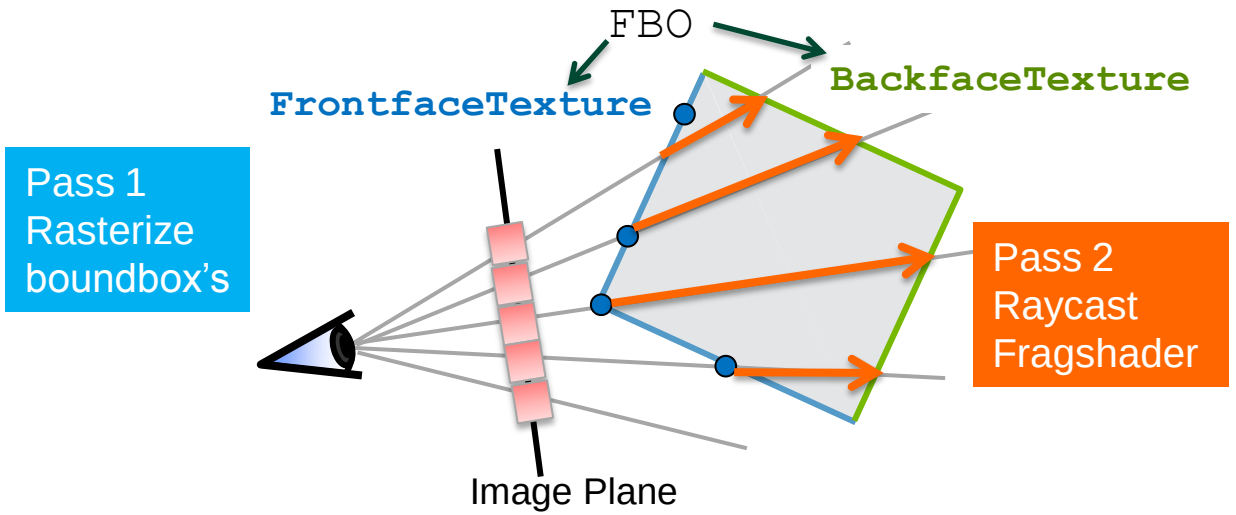
CUDA Raycasting - block Size vs fps



For a 512^3 volume,
Block size α 1/#blocks
#blocks maps to #cores



Multi-pass raycasting - OpenGL



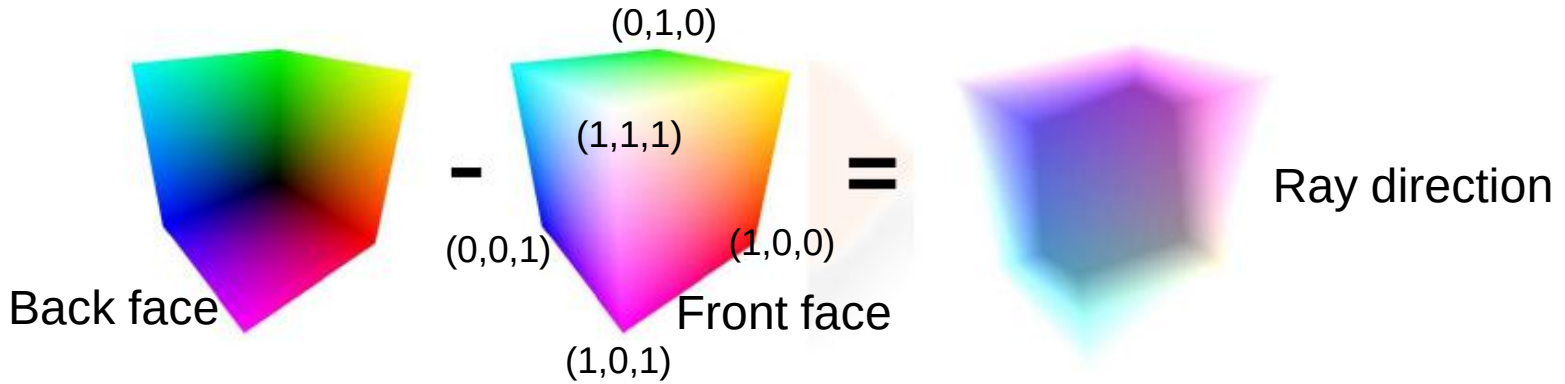
Init offscreen FBO and the 2 textures

```
glGenFramebuffersEXT(1, &FBO);
glBindFramebufferEXT(GL_FRAMEBUFFER_EXT, FBO);
glGenTextures(1, &BackfaceTexture); //The texture where the backface of the volume bounding cube is rendered to
glBindTexture(GL_TEXTURE_2D, BackfaceTexture);
glTexImage2D(GL_TEXTURE_2D, 0,GL_RGBA16F_ARB, winWidth, winHeight, 0, GL_RGBA, GL_FLOAT, NULL);
glGenTextures(1, &FrontfaceTexture); //The texture that has the front face image
glBindTexture(GL_TEXTURE_2D, FrontfaceTexture);
glTexImage2D(GL_TEXTURE_2D, 0,GL_RGBA16F_ARB, winWidth, winHeight, 0, GL_RGBA, GL_FLOAT, NULL);
```

Pass 1 - Rasterize to offscreen FBO's

Ray start/end positions to FBO with cull enabled

```
glEnable(GL_CULL_FACE);
//Draw the backface box by culling GL_FRONT
glFramebufferTexture2DEXT(GL_FRAMEBUFFER_EXT, GL_COLOR_ATTACHMENT0_EXT, GL_TEXTURE_2D, m_nBackfaceTexture, 0);
glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT );
glCullFace(GL_FRONT);
drawBoundBox (0.0, 0.0, 0.0, 1.0,1.0, 1.0);

//Now draw the frontface to texture by culling GL_BACK
glFramebufferTexture2DEXT(GL_FRAMEBUFFER_EXT, GL_COLOR_ATTACHMENT0_EXT, GL_TEXTURE_2D, m_nFrontfaceTexture, 0);
glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT );
glCullFace(GL_BACK);
drawBoundBox(0.0,0.0,0.0,1.0,1.0, 1.0);
```

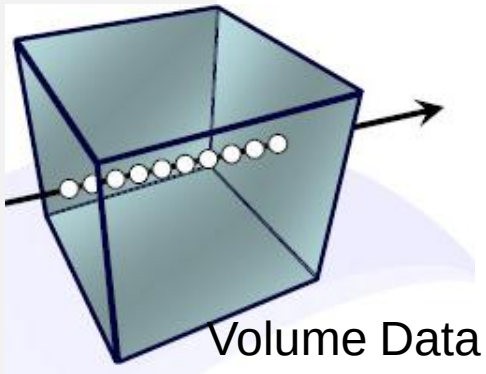


[Krugger and Westermann 2003]

Pass 2- Draw on-screen quad

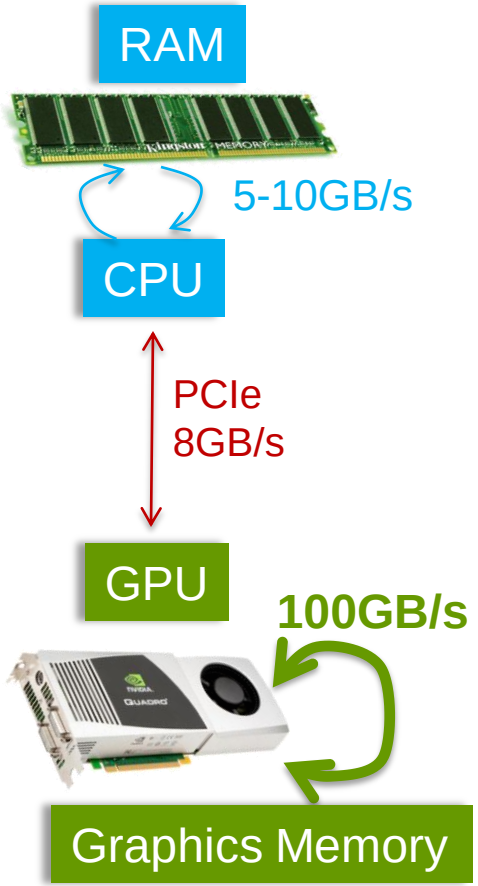
Invoke raycast fragment shader

```
uniform sampler3D volumeTex; //texture unit 0
uniform sampler1D transferFuncTex;  //texture unit 1
uniform sampler2D backfaceTex;  //texture unit 2
uniform sampler2D frontfaceTex;  //texture unit 3
uniform float stepInc; //inc for each sample
void main() { //back to front ray accumulation
      vec3 startPos = texture2D(frontfaceTex, gl_TexCoord[0].st).rgb;
      vec3 endPos = texture2D(backfaceTex, gl_TexCoord[0].st).rgb;
      vec3 ray = endPos - startPos;
      float rayLength = length(ray);
      vec3 step = stepInc*normalize(ray);
      for (int i=0; i<maxSamples; i++) {//back to front
            src = texture1D(transferFuncTex, texture3D(volumeTex, endPos-curSamplePos).a);
            dest.rgb = mix(dest.rgb, src.rgb, src.a); //(1- src_alpha)*dest_col + src_alpha*src_col
            dest.a = mix(dest.a, 1.0, src.a); //(1-src_alpha)*dest_alpha +    src_slpha
            if ((length(curSamplePos)>=rayLength))   //out of volume
                  break;
            if (dest.a>0.99) {
                  dest.a = 1.0;
                  break;
            }
            curSamplePos += step;
      } //end of for
      gl_FragColor = dest;
}
```



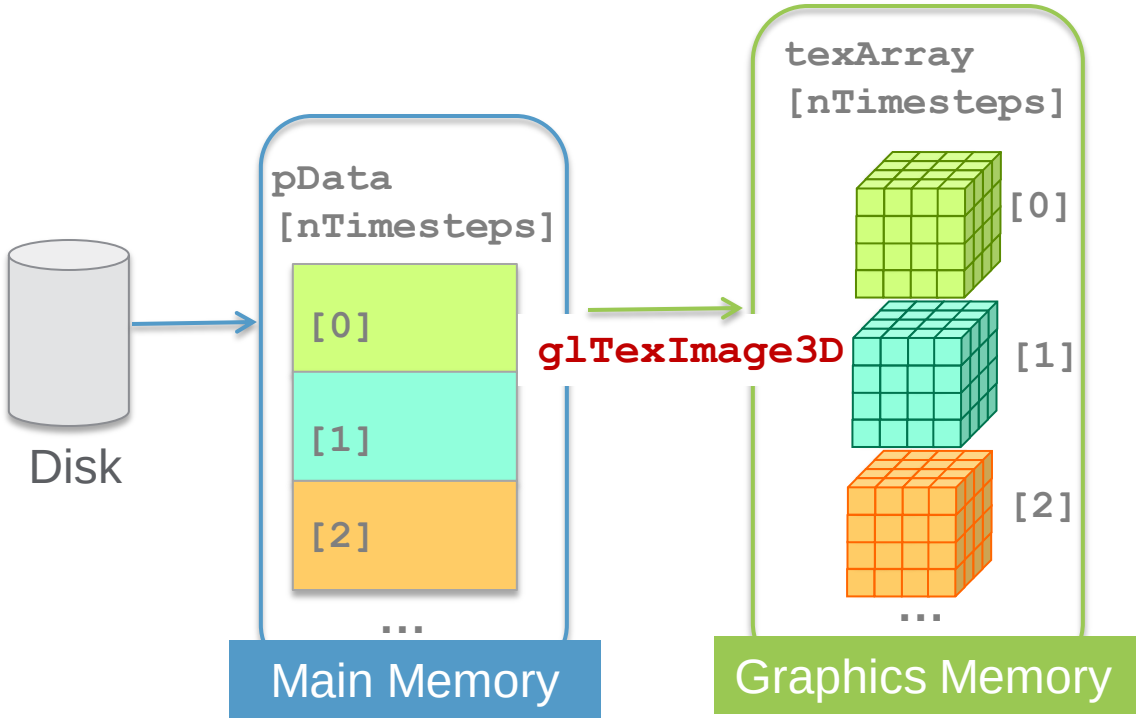
Extending to 4D - Single GPU

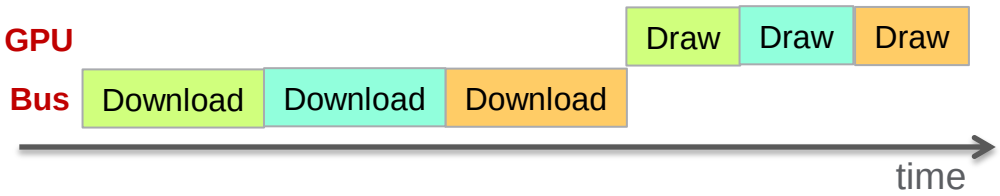
- Caching all data in graphics
- Streaming to graphics
- CPU asynchronous streaming
- GPU asynchronous streaming
- Comparisons



4D - Caching all time steps

- When graphics memory can hold entire 4D data
- Pros
  - As fast as draw
  - No data copy in RAM





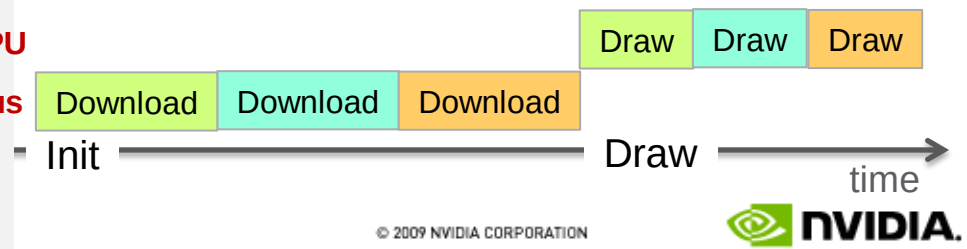
Caching - implementation

Init – allocate all textures and download data

```
glActiveTextureARB(GL_TEXTURE0_ARB);
Gluint* texArray = new GLuint[nTimesteps];
glGenTextures(nTimesteps,texArray);
//TODO - read from file to pData
for (int i=0;i < nTimesteps;i++) {//iterate over all textures and donwload
     glBindTexture(GL_TEXTURE_3D, texArray [i]);
     //TODO – Set Texture Params like wrap, filter using glTexParameteri
     //Download data to graphics
     glTexImage3D(GL_TEXTURE_3D,0,GL_INTENSITY,dimx,dimy,dimz,0,GL_LUMINANCE,GL_UNSIGNED_BYTE,pData[i]);
}
//TODO – Optional: delete pData from main memory
curTimeStep = 0; //initialize current time step
```

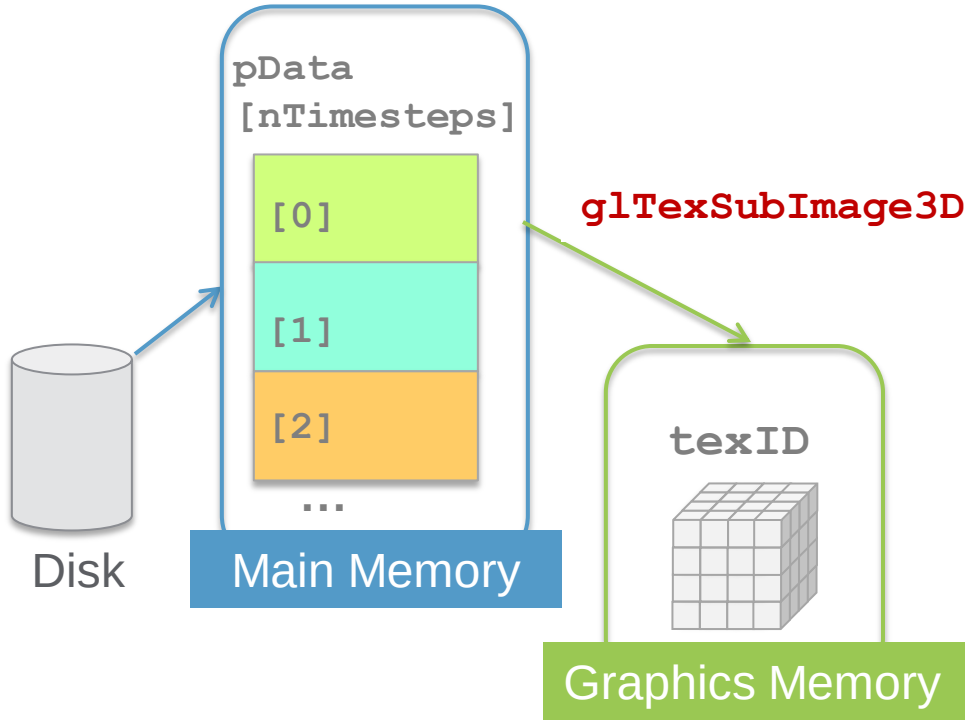
Draw – switch to right timestep

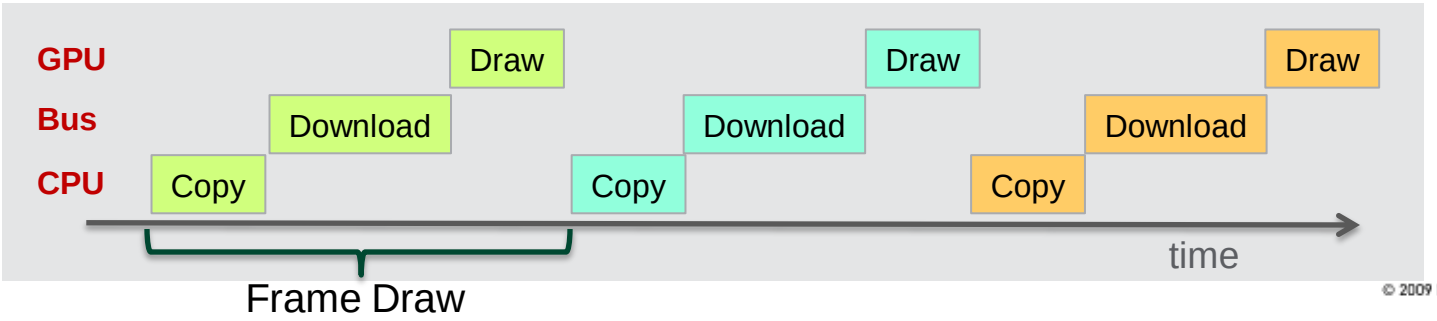
```
curTimeStep ++;
curTimeStep %= nTimesteps;
glActiveTextureARB(GL_TEXTURE0_ARB);
glBindTexture(GL_TEXTURE_3D, texArray[curTimeStep]);
//CALL DRAWING CODE HERE
```



Texture Streaming

- When graphics memory < 4D data
- Assume 4D data fits in RAM
- Download via glTexSub* involves CPU memcpy





Texture streaming

Init – allocate 1 texture and download 1st time step

```
glActiveTextureARB(GL_TEXTURE0_ARB);
glGenTextures(1,&texID);
//TODO - read from file to pData
glBindTexture(GL_TEXTURE_3D, texID);
//TODO - Set Texture Params like wrap, filter using glTexParameteri
//Download 1st time step to graphics
glTexImage3D(GL_TEXTURE_3D,0,GL_INTENSITY,dimx,dimy,dimz,0,GL_LUMINANCE,GL_UNSIGNED_BYTE,pData[0]);
curTimeStep = 0; //initialize current time step
```

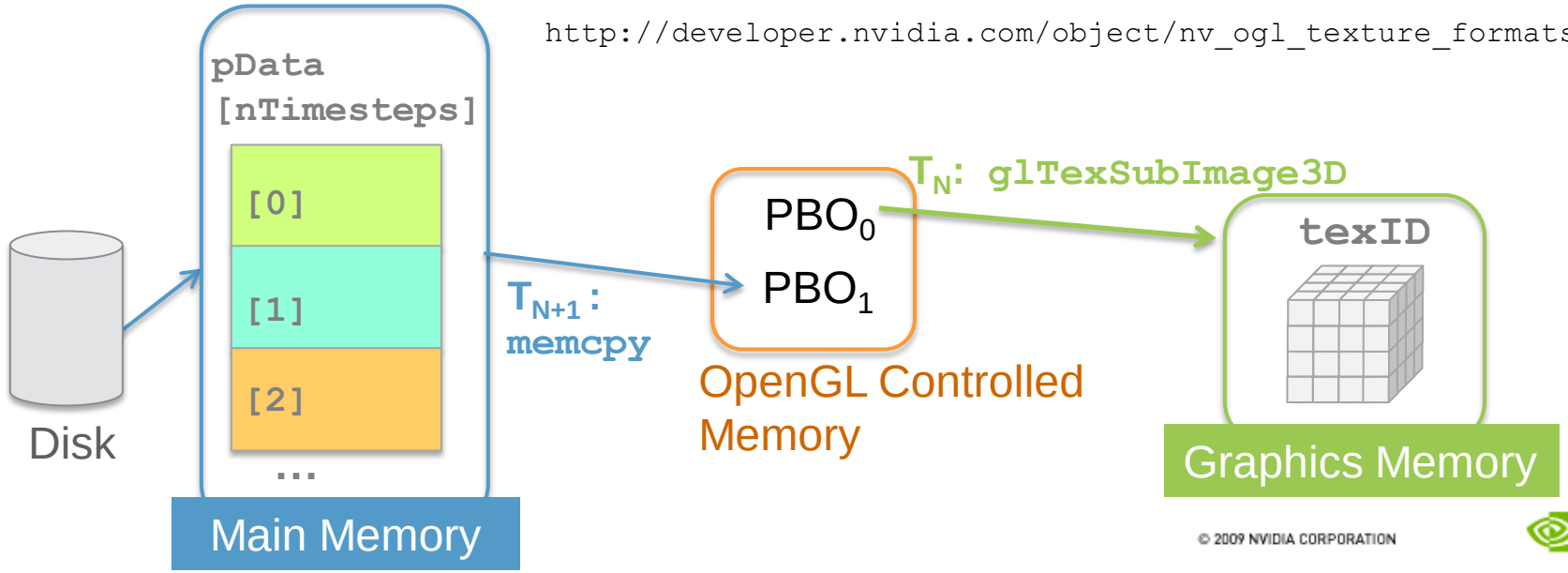
Draw – Download current time step volume to gfx

```
curTimeStep ++;
curTimeStep %= numTimesteps;
glActiveTextureARB(GL_TEXTURE0_ARB);
glBindTexture(GL_TEXTURE_3D, texID);
glTexSubImage3D(GL_TEXTURE_3D,0, 0,0,0,dimx,dimy,dimz,
          GL_LUMINANCE,GL_UNSIGNED_BYTE,m_pVolume[m_curTimeStep]);
//TODO - Call drawing code here
```

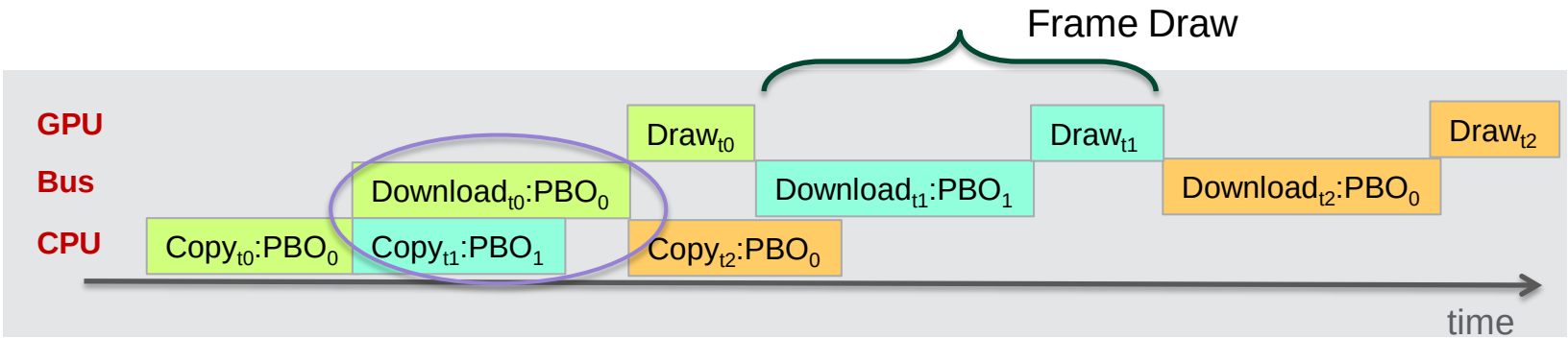
CPU Asynchronous streaming

- Non CPU-blocking transfer using Pixel Buffer Objects (PBO)
  - Ping-pong PBO's for optimal throughput
  - Data must be in GPU native format

http://developer.nvidia.com/object/nv_ogl_texture_formats.html



PBO Implementation



Init – allocate 2 PBO's and1 texture object

```
glGenBuffersARB(2, m_nPBOIds); //Allocate 2 PBO's
glBindBufferARB(GL_PIXEL_UNPACK_BUFFER_ARB, PBOIds[0]);
glBufferDataARB(GL_PIXEL_UNPACK_BUFFER_ARB,width*height*depth*sizeof(GLubyte),0,GL_STREAM_DRAW_ARB);
glBindBufferARB(GL_PIXEL_UNPACK_BUFFER_ARB, PBOIds[1]);
glBufferDataARB(GL_PIXEL_UNPACK_BUFFER_ARB,width*height*depth*sizeof(GLubyte),0,GL_STREAM_DRAW_ARB);
glBindBufferARB(GL_PIXEL_UNPACK_BUFFER_ARB, 0);

//TODO – Same texture initialization from "texture streaming" section
```

UNPACK - for download to GPU

PBO Implementation- Draw

Draw – app->pbo & pbo->tex transfer

```
m_curTimeStep ++;
m_curTimeStep %= m_pVolume.size();
static unsigned int curPBO = 0;
glBindTexture(GL_TEXTURE_3D,texId);
glBindBufferARB(GL_PIXEL_UNPACK_BUFFER_ARB, m_nPBOIds[curPBO]); //bind pbo
//Copy pixels from pbo to texture object
glTexSubImage3D(GL_TEXTURE_3D,0,0,0,0,xdim,ydim,zdim,GL_LUMINANCE,GL_UNSIGNED_BYTE,0);

//bind next pbo for app->pbo transfer
glBindBufferARB(GL_PIXEL_UNPACK_BUFFER_ARB, m_nPBOIds[1-curPBO]); //bind pbo
//to prevent sync issue in case GPU is till working with the data
glBufferDataARB(GL_PIXEL_UNPACK_BUFFER_ARB, xdim*ydim*zdim*sizeof(GLubyte), 0, GL_STREAM_DRAW_ARB);
GLubyte* ptr = (GLubyte*)glMapBufferARB(GL_PIXEL_UNPACK_BUFFER_ARB, GL_WRITE_ONLY_ARB);
assert(ptr);
memcpy(ptr,m_pVolume[m_curTimeStep],m_w*m_h*m_d);
glUnmapBufferARB(GL_PIXEL_UNPACK_BUFFER_ARB);
glBindBufferARB(GL_PIXEL_UNPACK_BUFFER_ARB,0);
curPBO = 1-curPBO;

//TODO - Call drawing code here
```

PBO to Texture

App to PBO

GPU Asynchronous

- Drawback of PBO's – GPU blocks, data download same thread as draw
- Fermi architecture - Copy Engine (CE) truly asynchronous downloads
- Quadro cards will have 2 CE's - overlap compute, download, readback
- Apps have a separate download/upload thread
- Driver manages synchronization and scheduling

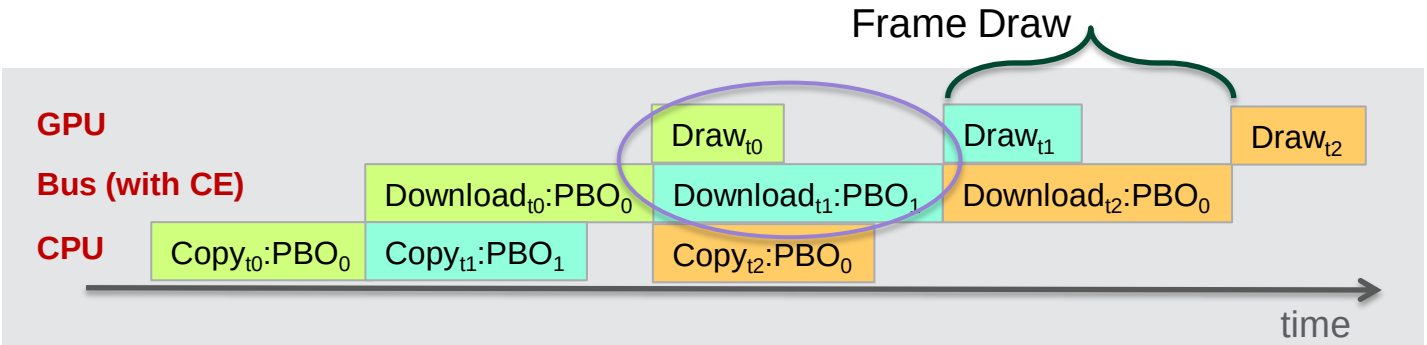
GPU Async - Implementation

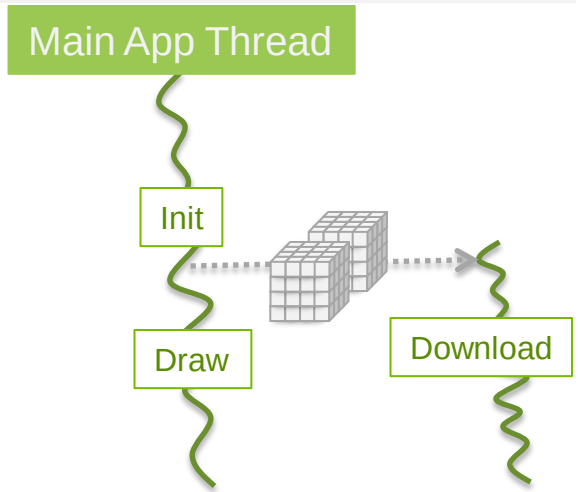
Main draw thread

```
//Get 2 OpenGL contexts from same DC
HGLRC downloadRC = wglCreateContext(hDC);
HGLRC drawRC = wglCreateContext(hDC);
//Before any loading, share textures between contexts
wglShareLists(downloadRC, drawRC); //Note order
//Create download thread from the main render thread
HANDLE downloadThread = CreateThread(NULL, NULL,
threadFunc, NULL, NULL, NULL);
while (!done) {
      wglMakeCurrent(hDC, drawRC);
      //Texture access and drawing does here
}
```

Download thread

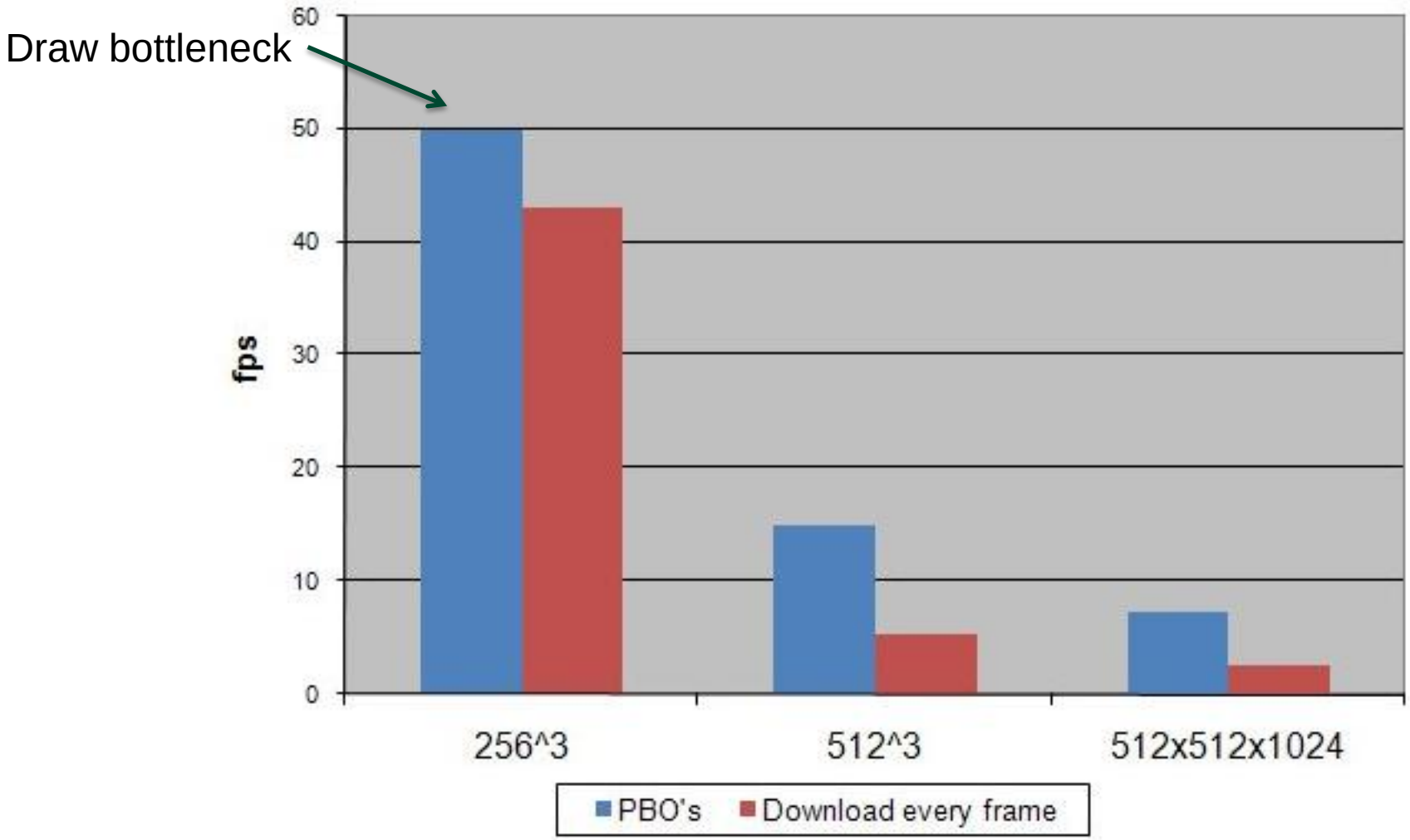
```
DWORD WINAPI threadFunc( LPVOID lpParam ) {
      wglMakeCurrent(hDC, downloadRC);
      glGenTextures(1,&texID);
      while (1) {
            //PBOs + glTexSubImage download as before

      } // while

      return TRUE;
}
```





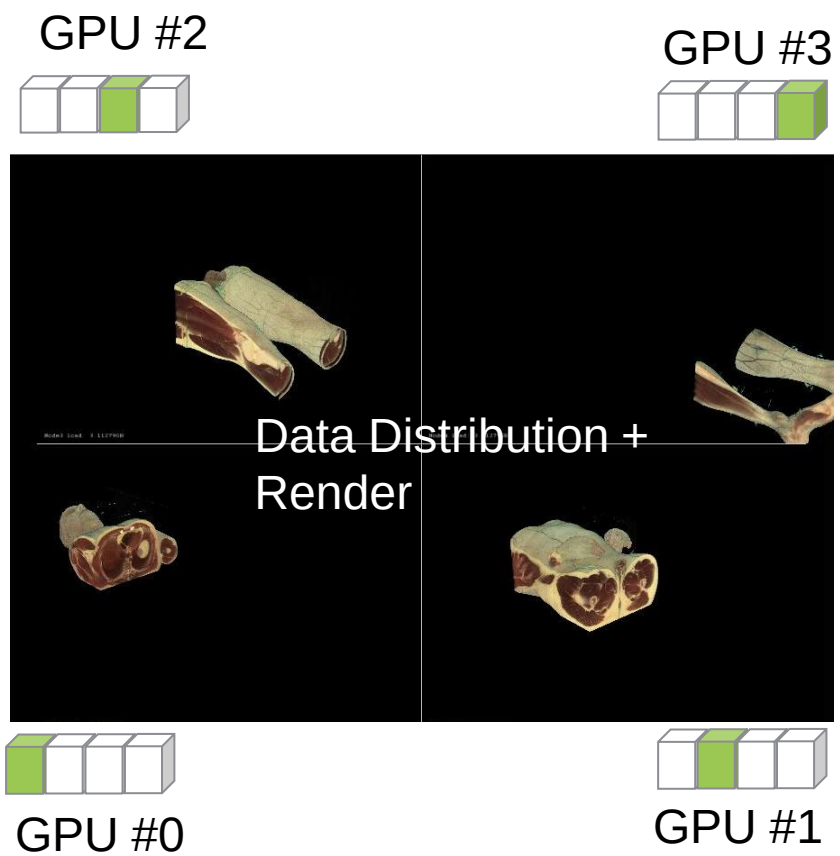
NVIDIA.

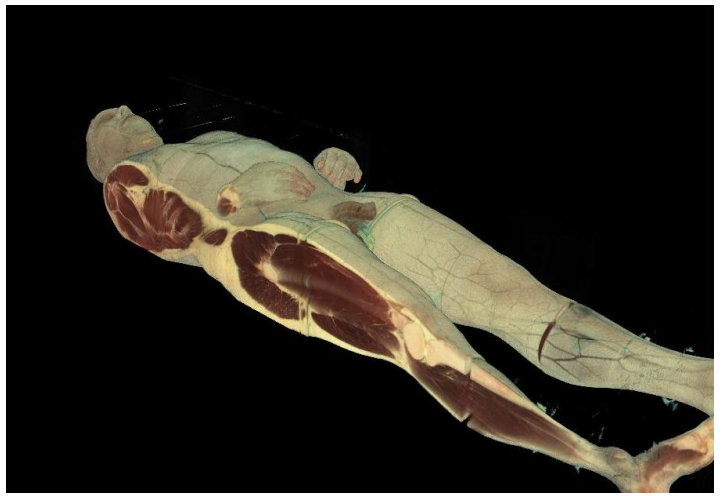
Comparisons



Multi-GPU Distributed Rendering

Sort-last rendering – Data distribution & Image Composition

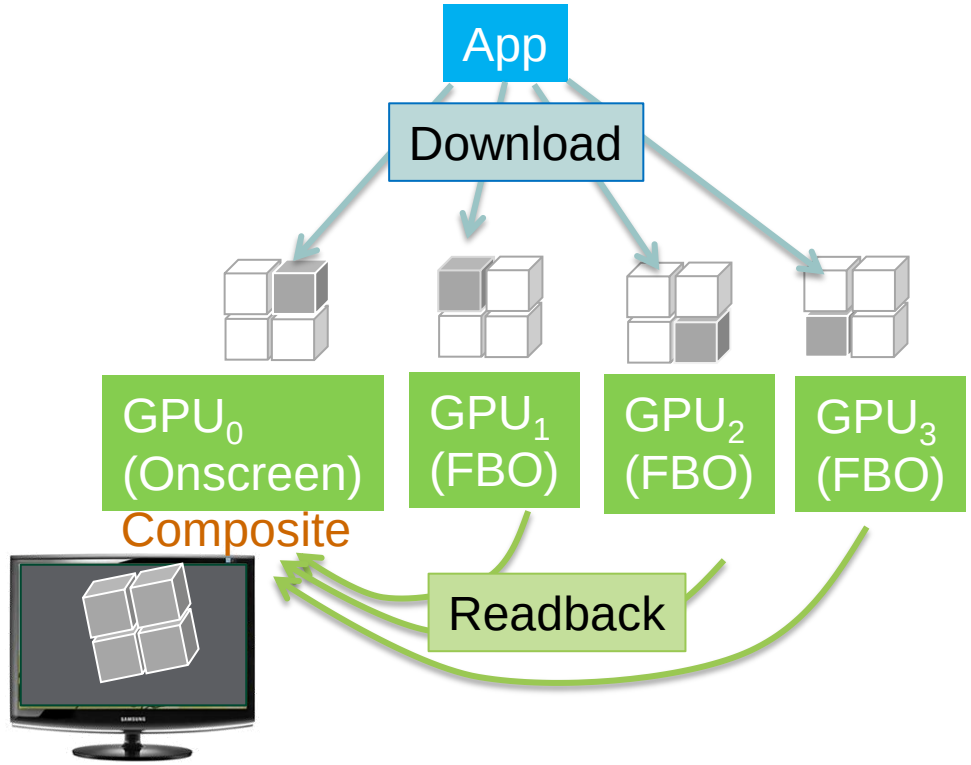




Alpha Composite

CompleX Middleware

- Part of Nvidia Application Acceleration Engine (AXE)
- APIs for
  - Volume/polygonal data distribution across multi-GPU
  - Composition with depth or alpha
- OpenGL based, quadroplex only
- On new fermi Quadro's, download+render+readback in parallel!



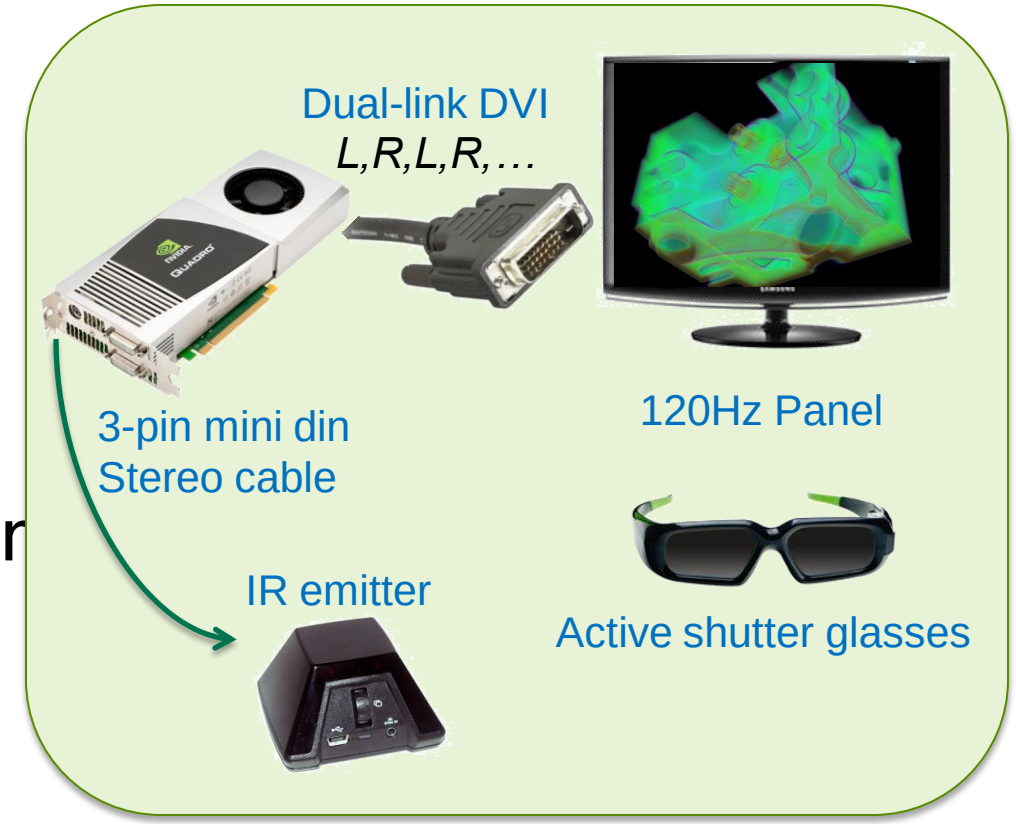
http://www.nvidia.com/object/complex.html

Stereo in Volume Rendering

- Provide depth cues not possible in 2D images
  - Binocular disparity specified by interocular distance to produce Left-Right images
- Volume rendering amorphous structures,
  - Spatial relationships, structure and scale and how they evolve with time in 4D simulations
- Critical for immersive environments – eg CAVE, Virtual endoscopy apps

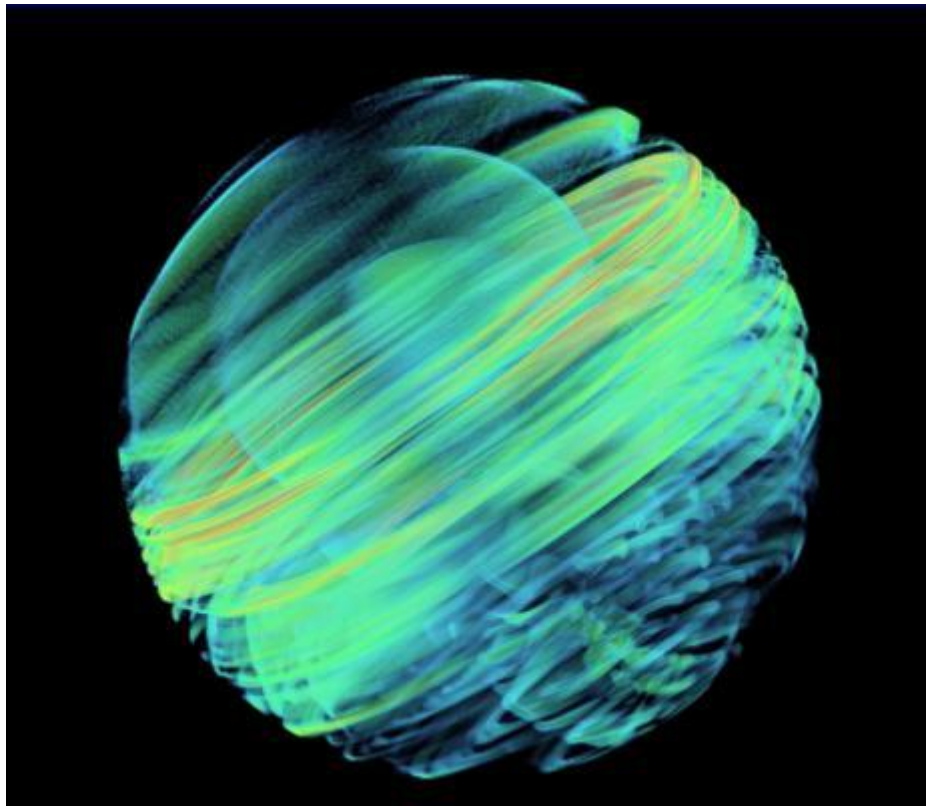
3D Vision stereo for Quadro

- Supports existing OpenGL Quadbuffer apps
- Works on Win XP, Vista
  - 185.xx drivers
  - Linux under dev
- Supported boards
  - Onboard stereo connector (FX 3700 and above)
  - USB on lower-end boards



http://www.nvidia.com/object/quadro_pro_graphics_boards.html

Demo and Q&A



shaliniv@nvidia.com

Seismic Wave propagation simulation
- 1994 Bolivia earthquake
Van Keken et al at Univ of Michigan
512x512x512x100 (12GB)