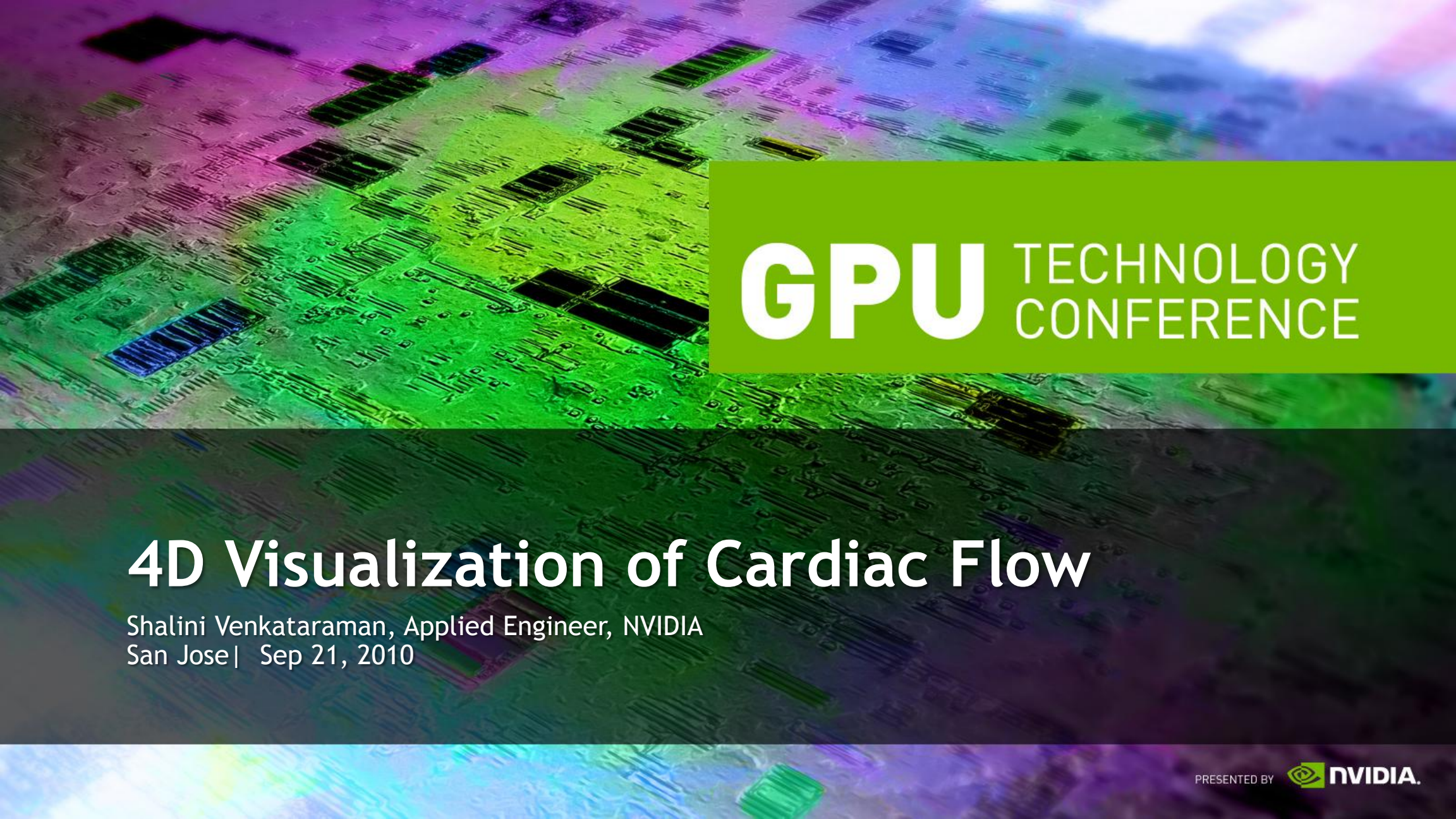


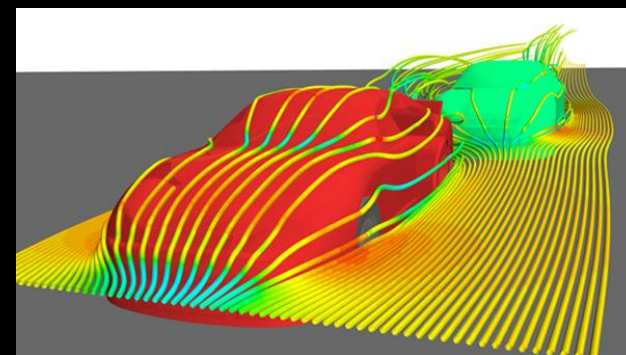
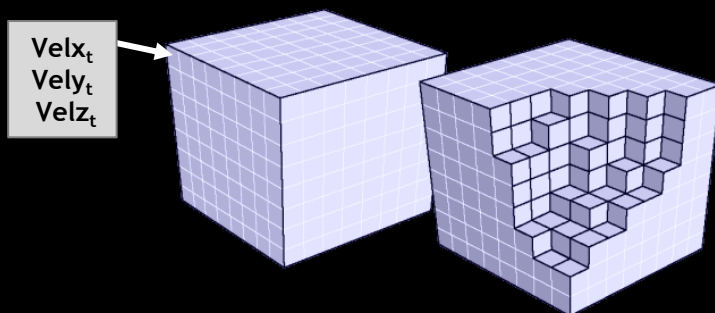


GPU TECHNOLOGY CONFERENCE

4D Visualization of Cardiac Flow

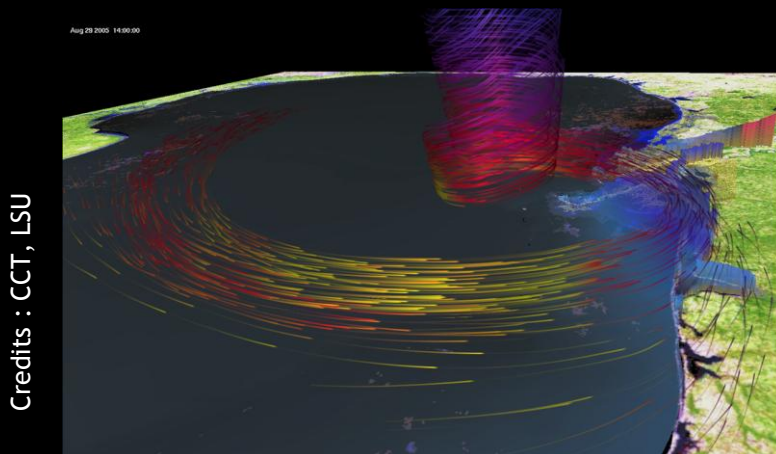
Shalini Venkataraman, Applied Engineer, NVIDIA
San Jose | Sep 21, 2010

Flow Visualization examples



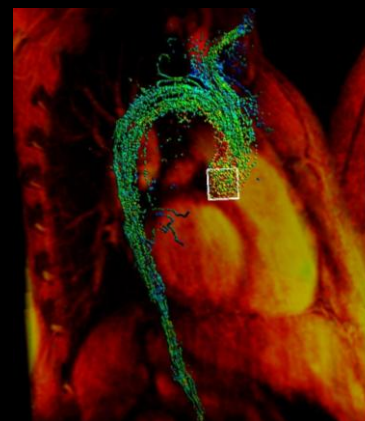
Credits : NASCAR

CFD Flow over objects



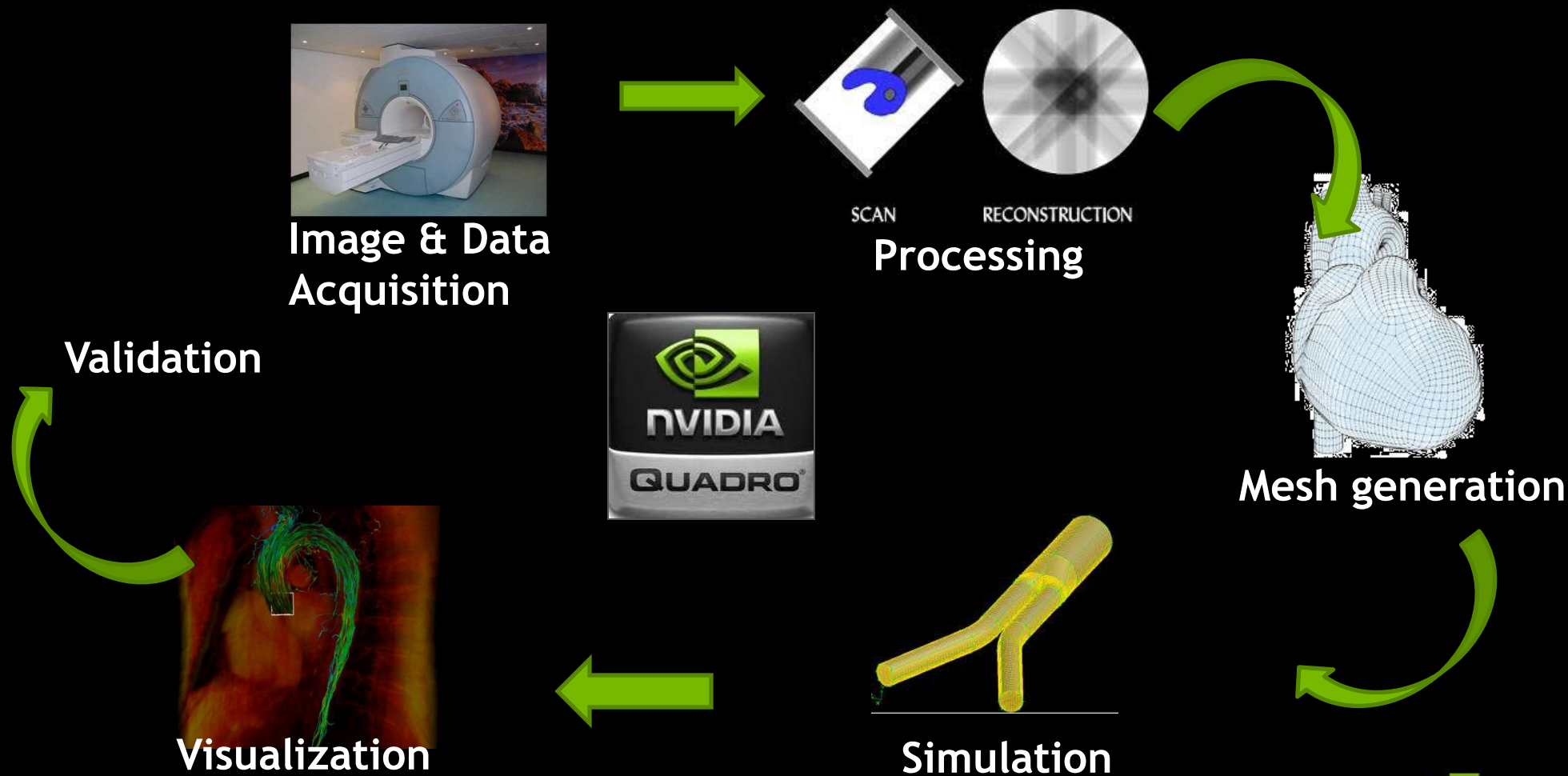
Credits : CCT, LSU

Atmospheric simulation



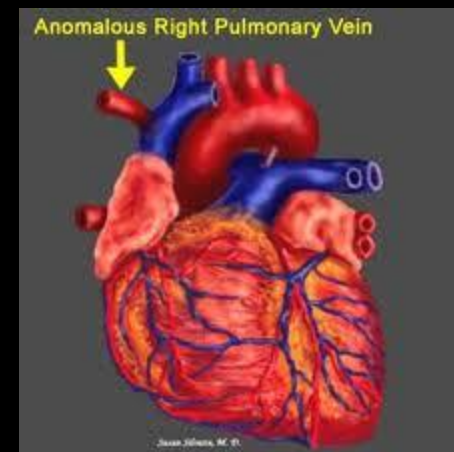
Medical - acquisition

Big picture - in Medicine



Specific Case Study - Heart Disease

- Congenital heart defects
 - Affects 1 in 125 newborns in the US
 - Structure causes abnormal flow patterns
- Pathology such as regurgitation, plaque change flow
- Challenges
 - Computation : Analyze 180 million vectors at 60fps
 - Memory : 3-8 GB



Cardiac flow visualization

- Non-invasive phase-contrast MRI to capture blood flow and stationary tissues in time
- Data size = $256 \times 256 \times 144 \times 20$ timesteps = 3.2GB
- Current workflow

Analyze

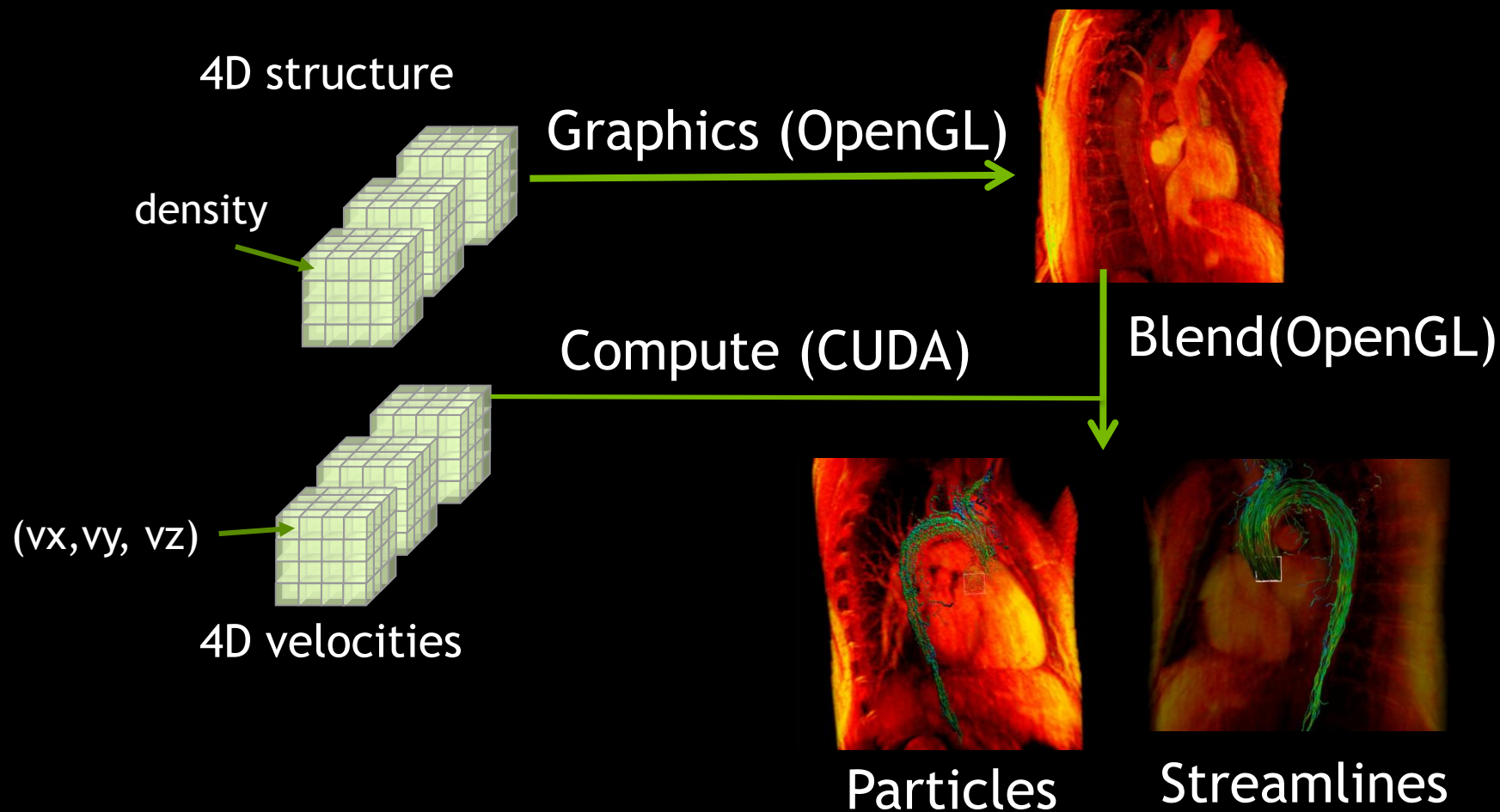


Visualize

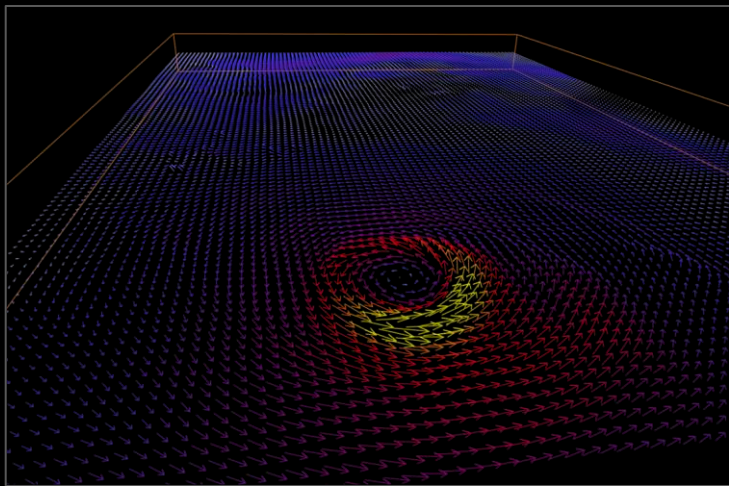
Render movies offline



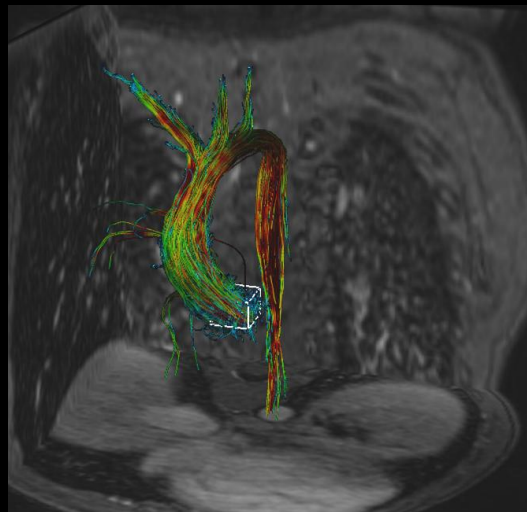
Compute & Render



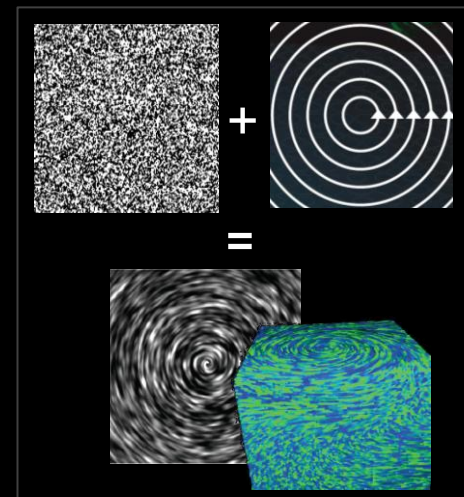
Local/Global Flow Characteristics



Local characteristics : arrow plots
+ Simple, no calculations
- Clutter



Global characteristics :
Particles, Streamlines
+ Where flow will go or comes from
- Depends on seed point



Texture based dense
Methods
+ Overall flow
- Compute intensive
(not covered)

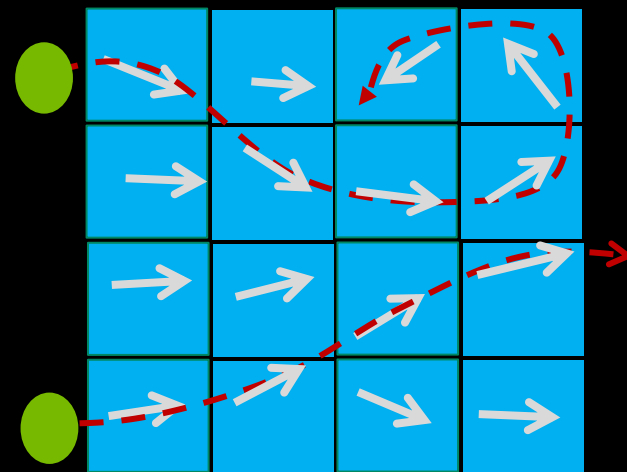
[Weiskopf2007]

GPU-based flow visualization

- Why use the GPU for flow visualization
 - Need for real-time manipulation of seed point
 - Analyze and viz at the same point.
- Traditional approaches use either
 - CPU (slow) or
 - GPGPU/Graphics shaders (awkward) and limitations such as geometry shader throughput

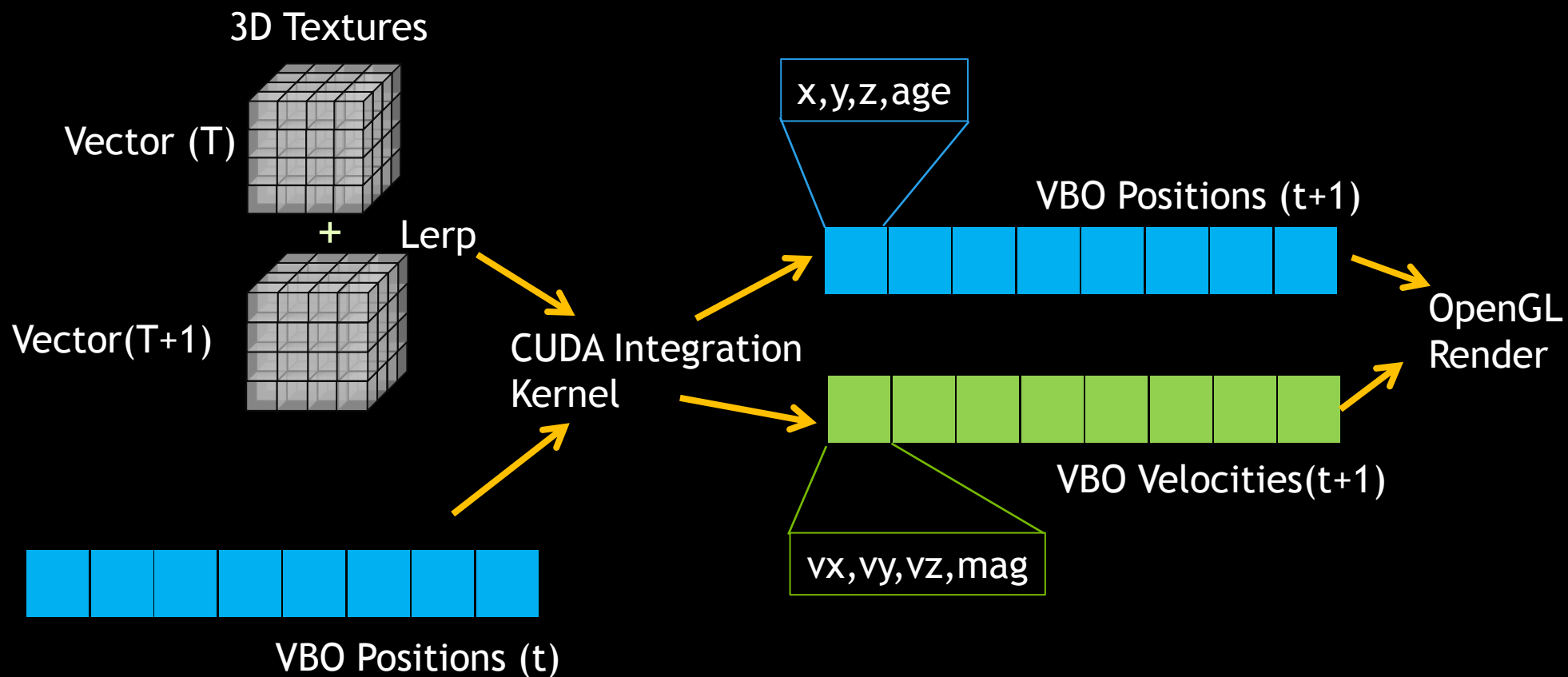
Particle Tracing

- GPU friendly - Particles move independent of each other
- Trace in parallel on GPU
- Repeat for 180 million points
- Sampling on discrete grid and interpolation
- Higher order interpolation -> Num ops and memory access



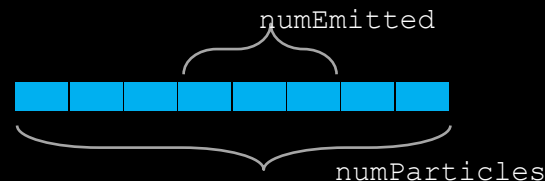
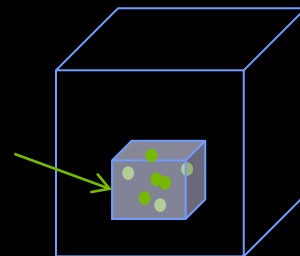
Tracing In 2D

GPU Particle tracing



Implementation

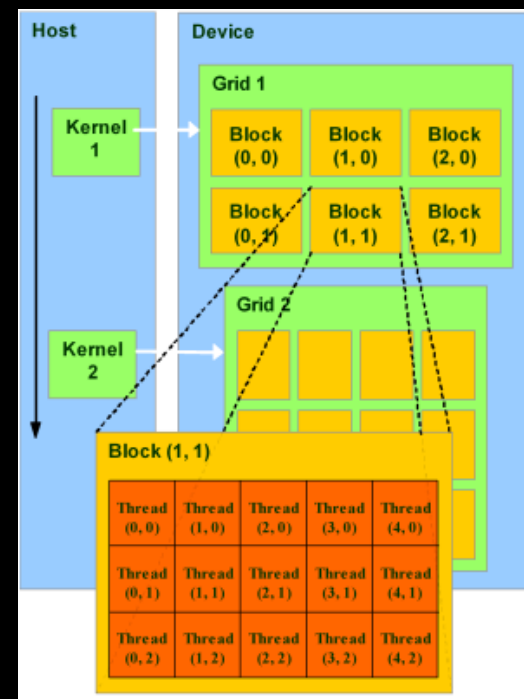
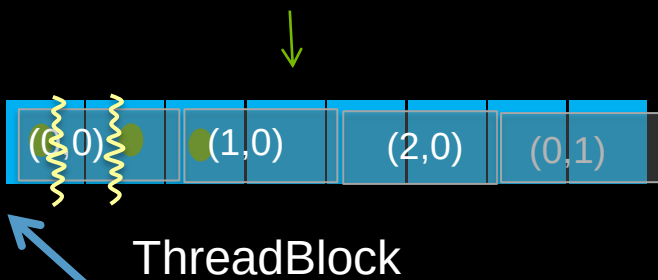
Emit random particles in ROI



Map Simulation time to Data Time

```
cudaBindTextureToArray (velTex)
cudaBindTextureToArray (velTexNext)
```

```
Vbo_velocity.map()
vbo_positions.map()
```



Implementation

Init

```
for (int i=0;i<NUMTimesteps;i++) {  
    readFromFile(i);  
    // SET VOLUMESIZE AND CHANNEL DESC  
    cudaMalloc3DArray(&d_velArray[i], &channelDesc, volumeSize);  
    // Copy data from host to cuda array in device  
    (d_velArray[i])  
    // SET COPYPARAMS FOR CUDAMEMCPY  
    cudaMemcpy3D(&copyParams);}  
  
cudaBindTextureToArray(velTex, d_velArray[0]);  
cudaBindTextureToArray(velTexNext, d_velArray[1]);
```

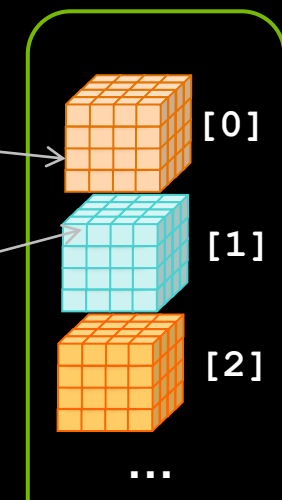
Compute

```
numThreads = min(blockSize, numParticles);  
numBlocks = iDivUp(numParticles, numThreads);  
integratedD<<< numBlocks, numThreads >>>(newPos, newVel,oldPos,  
oldVel,numParticles,deltaPhysicalTime
```

cudaArray *d_velArray
[Num]

Texture<..>
velTex

Texture<..>
velTexNext



CUDA
Memory

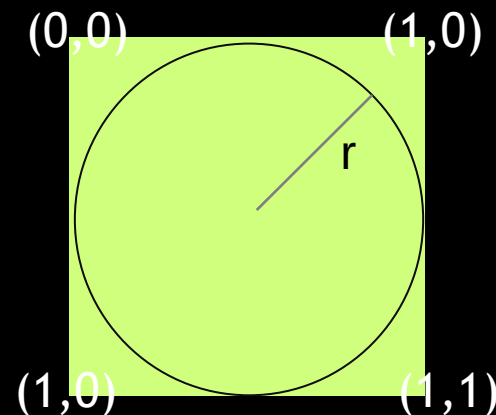
Rendering Particles

- Geometry

- Instancing possible but still lots of geometry
- Geometry shader bottlenecks

- Screen oriented lit sprites - `GL_POINT_SPRITE_ARB`

- Widely used in games [green2010]
- Vertex Shader : Calculate quad size based on distance to camera
- Fragment Shader
 - Calculate sphere normal and depth
 - Discard pixels outside circle



Implementation - vertex shader

```
uniform float pointRadius; // point size in world space
uniform float pointScale; // scale to calculate size in pixels
uniform vec4 eyePos;
varying float lifetime;
vmain()
{
    vec4 wpos = vec4(gl_Vertex.xyz, 1.0);
    gl_Position = gl_ModelViewProjectionMatrix * wpos;
    // calculate window-space point size
    vec4 eyeSpacePos = gl_ModelViewMatrix * wpos;
    vdist = length(eyeSpacePos.xyz);
    gl_PointSize = pointRadius * (pointScale / dist);

    gl_TexCoord[0] = gl_MultiTexCoord0; // sprite texcoord
    gl_TexCoord[1] = eyeSpacePos;

    gl_FrontColor = gl_Color;
    lifetime = gl_Vertex.w;
}
```

CUDA GPU Computing SDK - Smoke Particles Demo

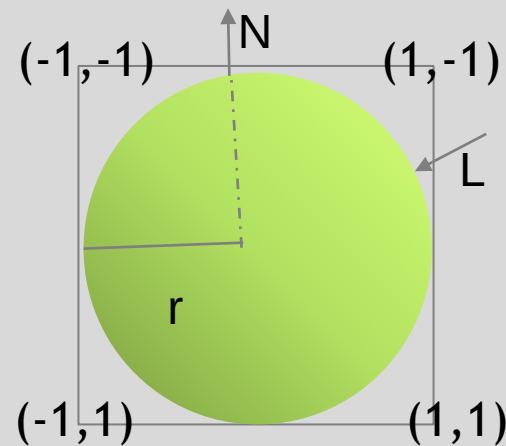
Implementation - Fragment Shader

```
uniform float pointRadius; // point size in world space
uniform sampler1D transferFuncTex; //texture unit 1
varying vec4 ep;
void main()
{
    const vec3 lightDir = vec3(0.577, 0.577, 0.577);
    // calculate normal from texture coordinates
    vec3 N;
    N.xy = gl_TexCoord[0].xy*vec2(2.0, -2.0) + vec2(-1.0, 1.0);
    float mag = dot(N.xy, N.xy);
    if (mag > 1.0) discard; // kill pixels outside circle
    N.z = sqrt(1.0-mag);

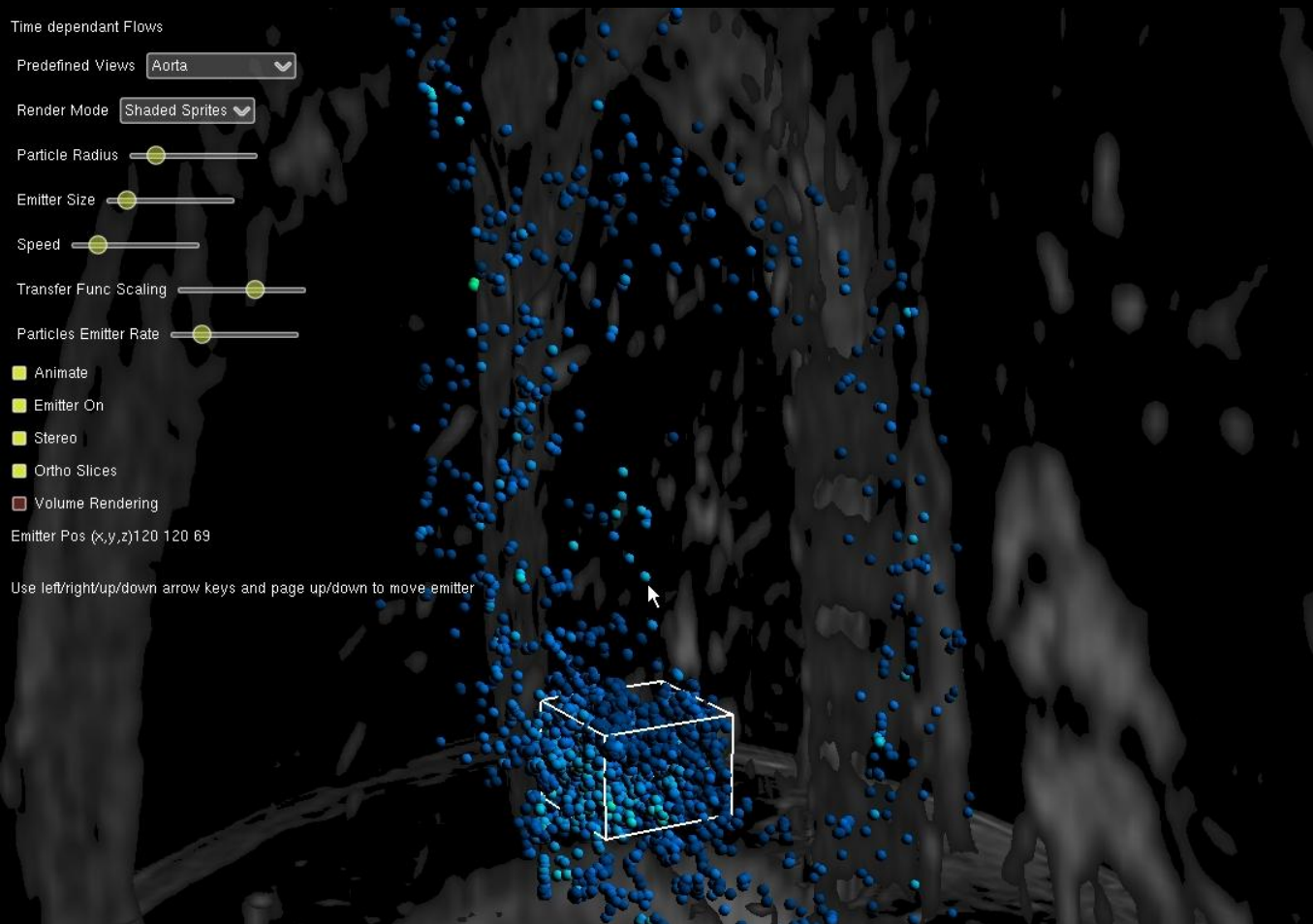
    // calculate depth
    vec4 eyeSpacePos = vec4(ep.xyz + N*pointRadius, 1.0); // position of this pixel on sphere in eye space
    vec4 clipSpacePos = gl_ProjectionMatrix * eyeSpacePos;
    gl_FragDepth = (clipSpacePos.z / clipSpacePos.w)*0.5+0.5;

    // calculate lighting
    float diffuse = max(0.0, dot(lightDir, N));

    //use gl_Color.a which has the velocity magnitude normalized and use that for
    //lookup in 1d texture for the real color
    vec4 color = texture1D(transferFuncTex, (gl_Color.a-offset)*scale);
    gl_FragColor = color*diffuse;
}
```



Results - Movie



16K particles at 275 fps

Streamlines & Pathlines

- Streamlines - Particle trajectory at fixed time step
- State of equilibrium or snapshot
- Pathlines - Varies with time, particle trajectory in unsteady flow
- Tracing massless particles

Streamline generation with CUDA

Streamline 0

Streamline 1

Streamline n-1



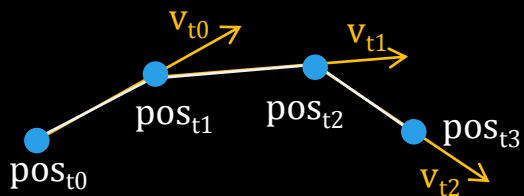
positionVBO[n*maxLength]
velocityVBO[n*maxLength]

length[n]

```
For each line
Get start pos in the vbo's = lineIdx*lineLength
Get currentLength for eachline from length array
//only if we are in the 2nd last element
{
    get current position[start+curlength]
    getcurrentvel[start+curlength]
    transfer to physical domain
    euler integration
    normalize back to texcoords 0..1
    setnextpositioon[start+curlength+1]
    //from the position, extract the velocity from the 3d
    textures and save it
    setnextvelocity
}
```

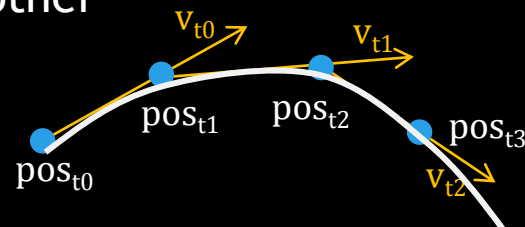
Integration Methods

Euler - Simple & fast but inaccurate



```
pos += (vel*deltaTime); //simple euler  
integration
```

Runge Kutta 4 - more lookups
but smoother



```
float3 k1 = make_float3(vel.x,vel.y,vel.z)*deltaTime;  
float3 newpos = pos + (k1 / 2.0f);  
float4 newvel = tex3D(velTex, newpos.x, newpos.y, newpos.z);  
  
float3 k2 = make_float3(newvel.x,newvel.y,newvel.z)*deltaTime;  
newpos = pos + (k2 / 2.0f);  
newvel = tex3D(velTex, newpos.x, newpos.y, newpos.z);  
  
float3 k3 = make_float3(newvel.x,newvel.y,newvel.z)*deltaTime;  
newpos = pos+k3;  
newvel = tex3D(velTex, newpos.x, newpos.y, newpos.z);  
  
float3 k4 = make_float3(newvel.x,newvel.y,newvel.z)*deltaTime;  
pos += ((k1 / 6.0f) + (k2 / 3.0f) + (k3 / 3.0f) + (k4 / 6.0f));
```

Rendering Streamlines

■ GL_LINE_STRIP

vbo_positions -> vertex array

vbo_velocity -> color array

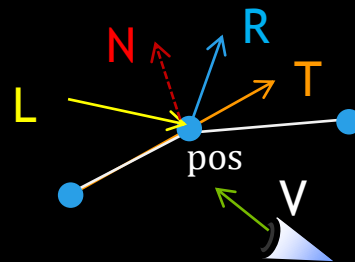
```
glBindBufferARB(GL_ARRAY_BUFFER_ARB,
mPosVboStreamlines);
glVertexPointer(4, GL_FLOAT, 0, 0);
glEnableClientState(GL_VERTEX_ARRAY);

glBindBufferARB(GL_ARRAY_BUFFER_ARB,
mVelVboStreamlines);
glColorPointer(4, GL_FLOAT, 0, 0);
glEnableClientState(GL_COLOR_ARRAY);

for (int i=0;i<m_numLines;i++)
    glDrawArrays(GL_LINE_STRIP, start, length[i]);
```

■ Illumination [Zockler1996]

- Lines have no distinct normal
- Use coplanar property:
Normal+Reflectance,
Tangent(vector) & Light!



$$I = K_a + K_d L \cdot N + k_s (V \cdot R)^n$$

$$L \cdot N = \sqrt{1 - (L \cdot T)^2}$$

$$V \cdot R = \sqrt{1 - (L \cdot T)^2} * \sqrt{1 - (V \cdot T)^2}$$

Illuminated Streamlines shader

```
varying vec4 color;
varying vec4 specular;
{
    vec3 view = normalize(cameraPosition.xyz - gl_Vertex.xyz);
    vec3 tangent = 2.0*gl_Color.rgb - 1.0; //normalize to -1..1
    float VdotT = abs(dot(view, tangent));

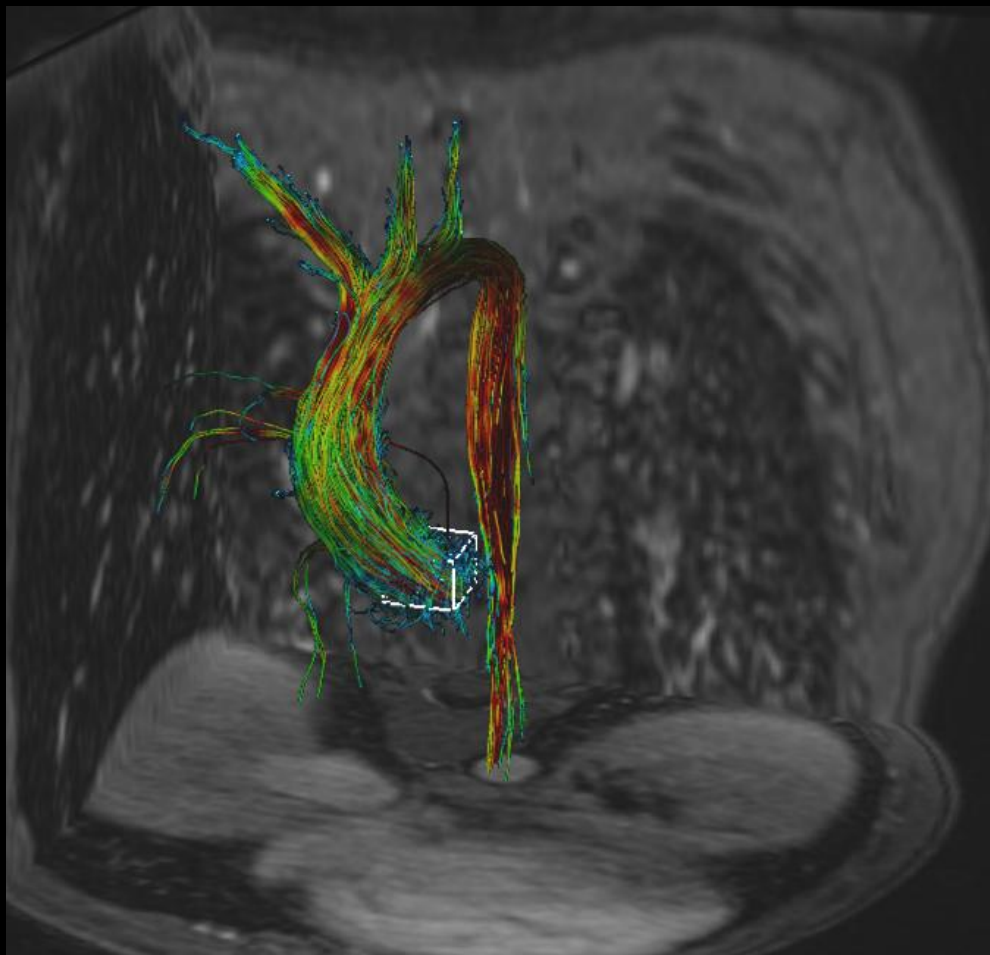
    const vec3 light = vec3(0.577, 0.577, 0.577); //light direction is constant same as for
point spheres case
    float LdotT = abs(dot(light, tangent));

    float LdotN = sqrt(1.0 - LdotT *LdotT );
    float VdotR = sqrt(1.0 - VdotT *VdotT) * sqrt(1.0 - LdotT *LdotT)

    color = ambient + diffuseFactor*LdotN
    specular = specularFactor*pow(VdotR, shininess);

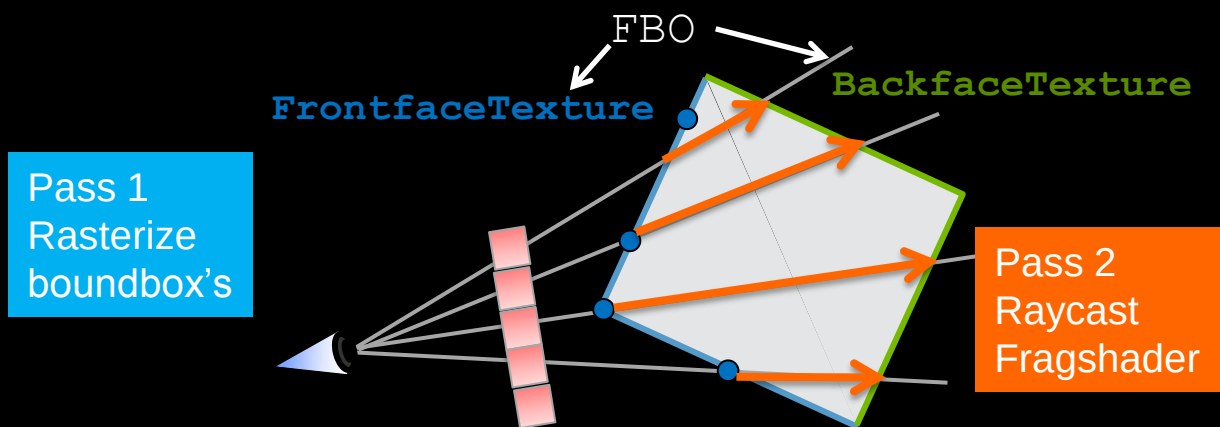
    gl_TexCoord[0] = gl_MultiTexCoord0;
    gl_Position = gl_ModelViewProjectionMatrix * vec4(gl_Vertex.xyz, 1.0);
};
```

Results - Movie



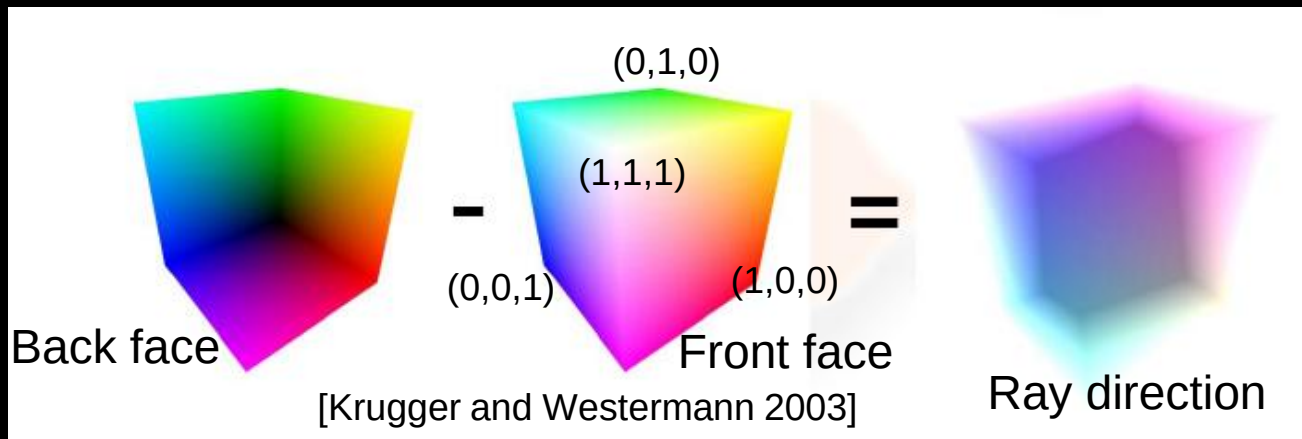
Volumetric rendering

- Full 3d context for the flows with slicing
- Using raycasting in OpenGL fragment shader



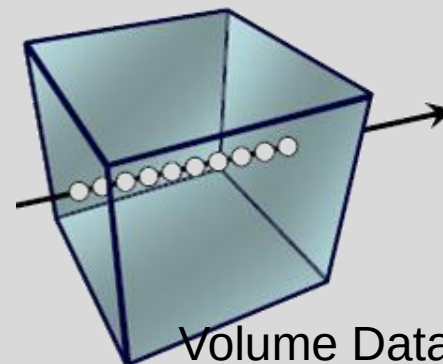
Pass 1 - Rasterize to offscreen FBO's

```
glEnable(GL_CULL_FACE);  
//Draw the backface box by culling GL_FRONT  
glFramebufferTexture2D(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);  
drawBoundingBox (0.0, 0.0, 0.0, 1.0,1.0, 1.0);  
  
//Now draw the frontface to texture by culling GL_BACK  
glFramebufferTexture2D(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);  
drawBoundingBox (0.0,0.0,0.0,1.0,1.0, 1.0);
```



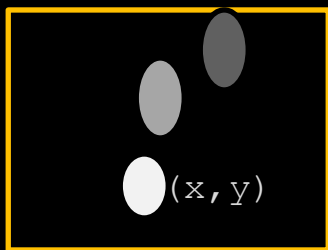
Pass 2 - Draw on-screen quad

```
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_alpha
        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;
}
```

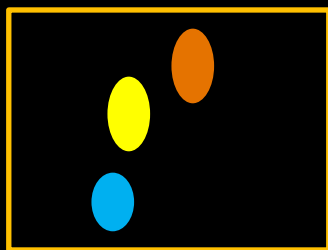


Blending Geometry and Volumes

1. Render opaque particles



Depth Texture

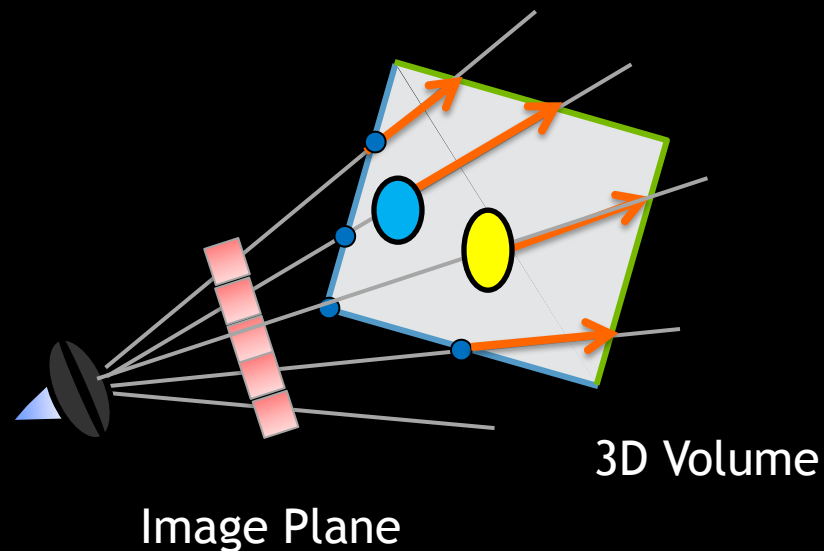


Color Texture

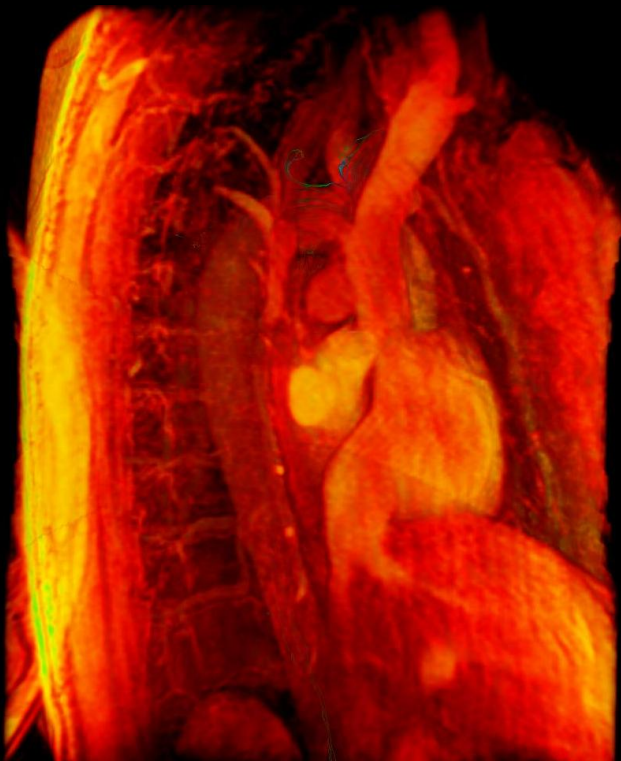
Transform to volume space

```
vec4 depthViewPos = (x,y,depthTexture(x,y),1.0)
//normalize to [-1,1]^3
//back project to volume space
inverse(gl_ModelViewProjectionMatrix)*depthViewPos
```

2. Raycast front-to-back translucent volumes

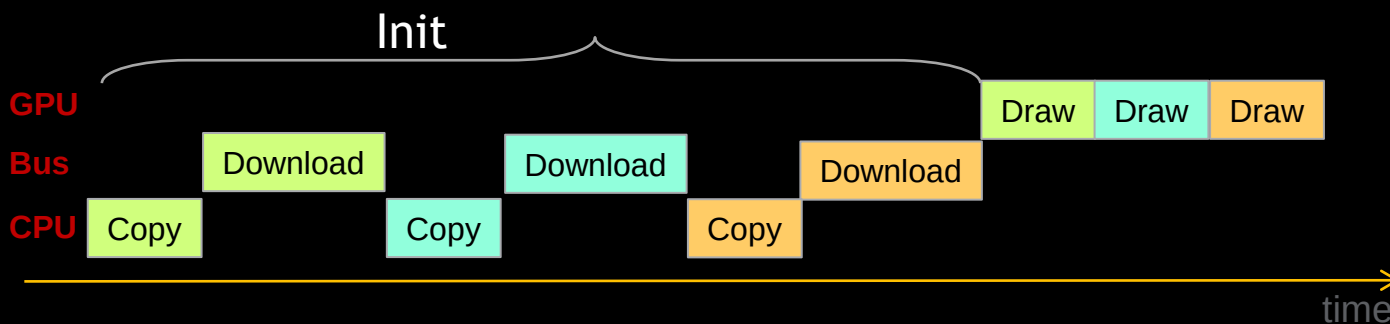
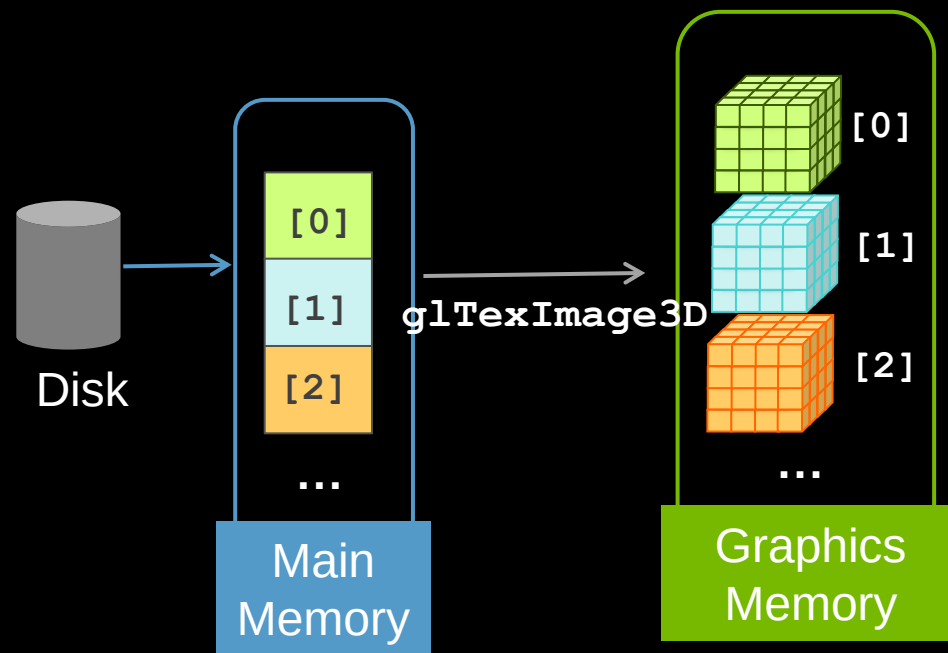


Results - movie



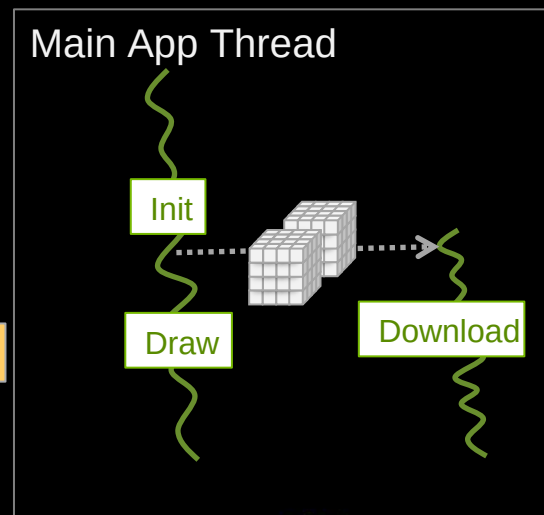
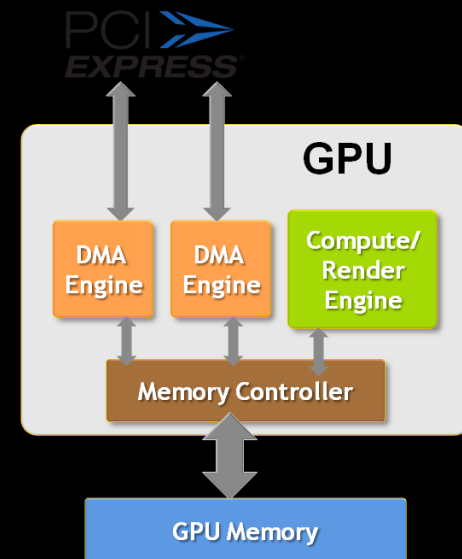
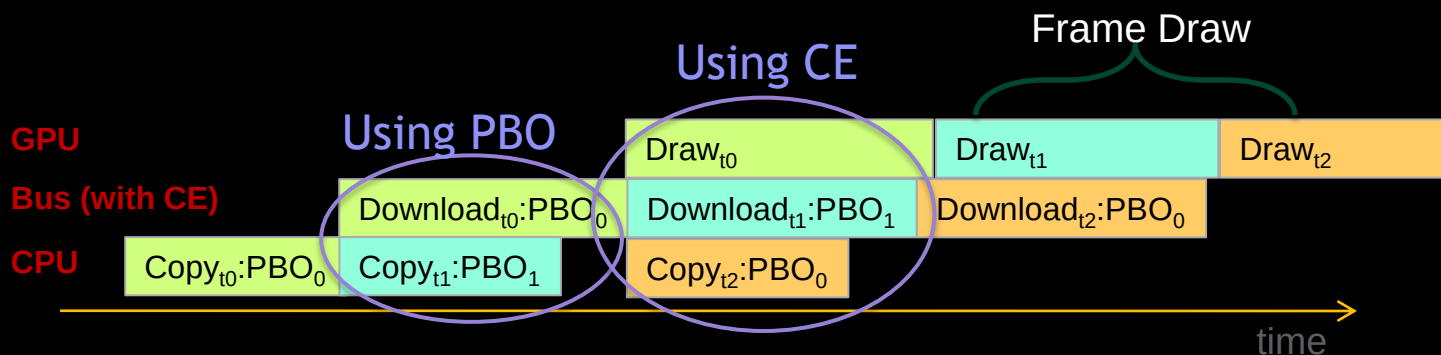
4D Volume Rendering - Cached

- + As fast as draw
- - Needs lot of graphics memory



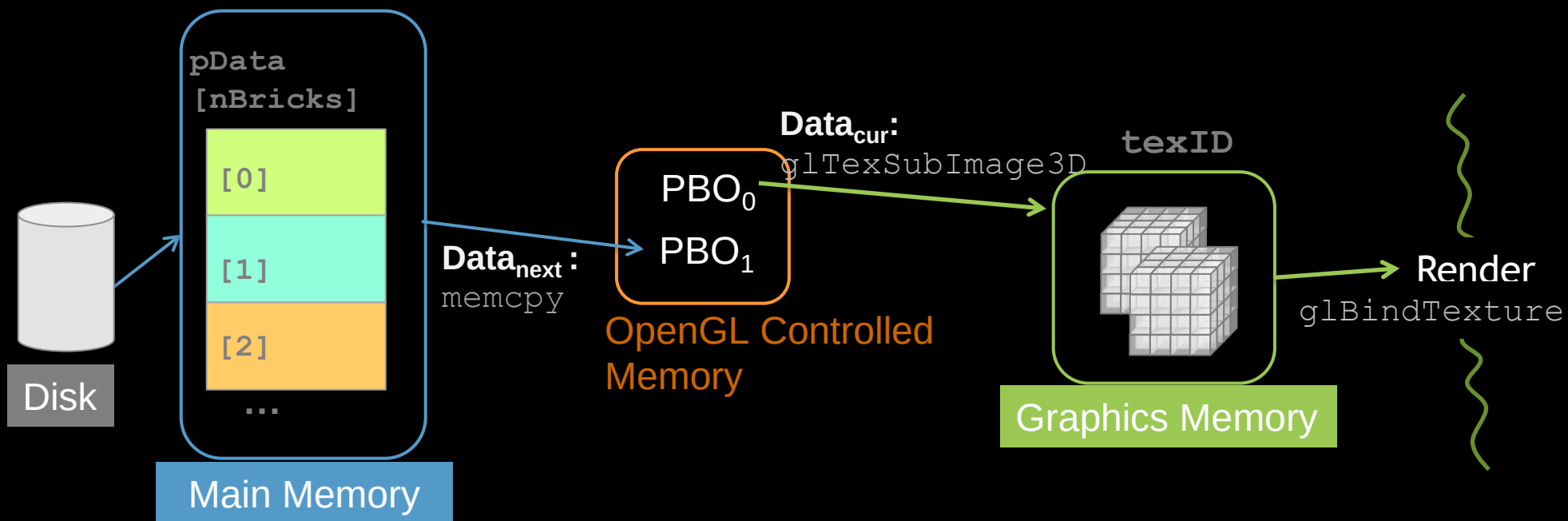
GPU Asynchronous Streaming

- Flow data is huge
 - 4x volume data + time
- Quadro dual copy engines
 - Overlap download, render & readback
 - + Small memory footprint
 - - Limited by PCI bandwidth



Implementation

- Using OpenGL PBOs in the download thread
- ARB_SYNC used for context synchronization



Movie

Time dependant Flows

Predefined Views Aorta

Render Mode Streamlines

Line Width

Emitter Size

Speed

Raycasting Step Size

Transfer Func Scaling

Streamlines Emitter Rate

☒ Animate

☒ Emitter On

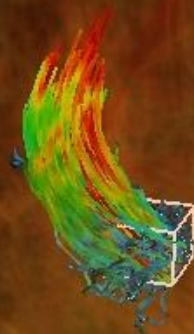
☒ Stereo

☐ Ortho Slices

☒ Volume Rendering

Emitter Pos (x,y,z) 120 120 69

Use left/right/up/down arrow keys and page up/down to move emitter



Need for Stereo

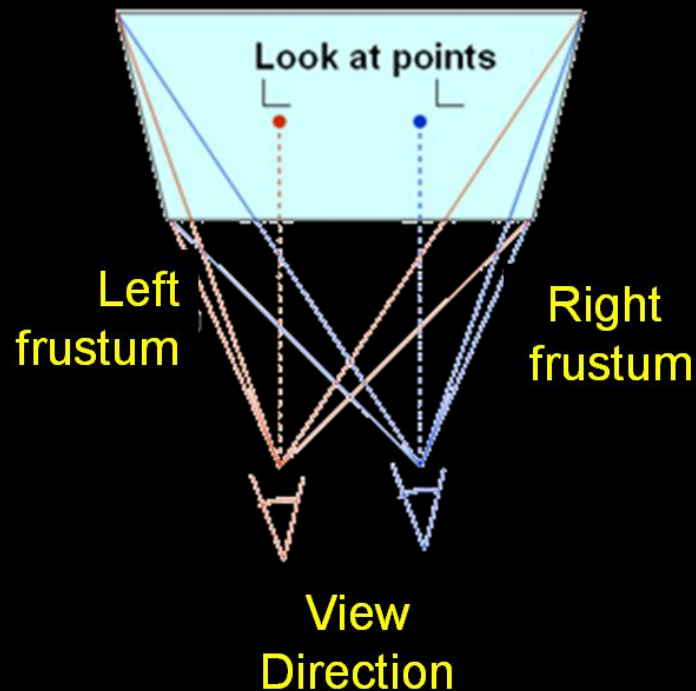
- Provide depth cues not possible in 2D images
 - Binocular disparity specified by interocular distance to produce Left-Right images
- Volume rendering amorphous structures with opaque geometry
 - Spatial relationships, structure and scale and how they evolve with time in 4D simulations
- Critical for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

Implementation

```
double IOD = 0.5; //intraocular distance
void setFrustum(void) {
    //sets top of frustum based on fovy and near clipping plane
    double top = nearZ*tan(DTR*fovy/2);
    double right = aspect*top; //sets frustum right based on aspect ratio
    double frustumshift = (IOD/2)*nearZ/screenZ;
    leftCam.topfrustum = top;
    leftCam.bottomfrustum = -top;
    leftCam.leftfrustum = -right + frustumshift;
    leftCam.rightfrustum = right + frustumshift;
    leftCam.modeltranslation = IOD/2;

    rightCam.topfrustum = top;
    rightCam.bottomfrustum = -top;
    rightCam.leftfrustum = -right - frustumshift;
    rightCam.rightfrustum = right - frustumshift;
    rightCam.modeltranslation = -IOD/2;
}

void draw(void) {
    glDrawBuffer(GL_BACK_LEFT); //draw to back left buffer
    //... Load left cam frustum(GL_PROJECTION)
    //.. model translation (GL_MODELVIEW)
    glDrawBuffer(GL_BACK_RIGHT); //draw to back right buffer
    //... Load right cam frustum (GL_PROJECTION)
    //.. and model translation(GL_MODELVIEW)
}
```



NVIDIA 3D Vision Pro

- With existing quadbuffered OpenGL apps
- WinXP/Vista/Linux
- Supported boards
 - Onboard stereo connector (3700+)
 - USB on lower-end boards



Future Work

- Backtrace - given a point, find out the particles that contributed to it
- Importance driven density adjustment like vorticity, divergence etc, mapping vector values -> scalar
- Integrate within clinical environment, Stanford pediatric and adult hospital, connecting to PACS system etc, medical specific GUI

Demo & QA

