GPU for Game Computing

Avi Bleiweiss

Overview

- GPU transition
  - CISC to RISC
- Dynamic architecture
  - Compute, bandwidth
- Data parallel
  - Often algorithm redesign
- *Game computing*
  - Simulation, AI, Audio

Outline

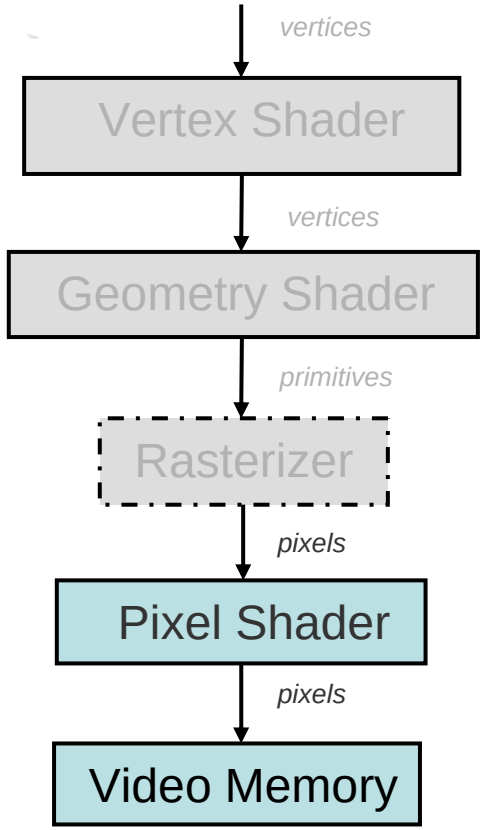
- GPU, processor array
- Compute abstraction
- Physics simulation
- Mesh video mapping
- Results, summary
  - FYSI, physics simulation case study

Outline

- GPU, processor array
- Compute abstraction
- Physics simulation
- Mesh video mapping
- Results, summary

Pipeline

- Deeply programmable
  - Vertex, geometry, pixel
- Precise and flexible
  - 32 bit IEEE float
- Unified shader model
- Compute pipeline
  - Simplified modality, API



Arithmetic

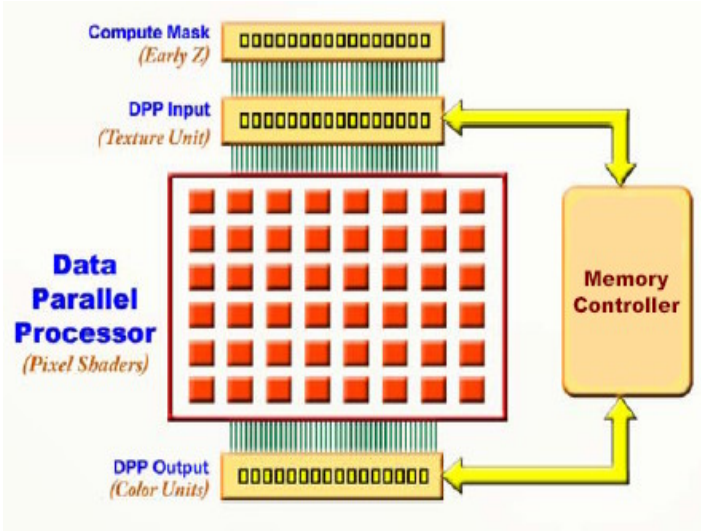
- Raw compute power
  - 48 pixel engines, each
  - Executes 4 float mad per cycle
  - @650 MHz -> **250 GFlops**
- Shader model 3.0
  - Dynamic branching
- Shader model 4.0
  - Integer arithmetic

Bandwidth

- Memory
  - 256 bits wide, double edge
  - @775 MHz -> **50 GBytes/sec**
  - 0.5 GByte video memory
- I/O, PCI Express x16
  - **4 GBytes/sec**
  - Multi-GPU shared resources
- Relatively small caches

Data Concurrency

- Grid formation
  - ALU heavy architecture
- Stream data memory system
- Many in flight threads
  - Hide memory latency
- No static data
- No read-modify-write buffers



Scatter/Gather

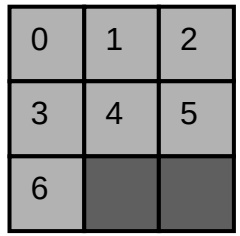
- Scatter
  - Indirect memory write (*data[i] = x*)
  - Populate data structures
  - Usually performed on CPU
- Gather
  - Indirect memory read (*x = data[i]*)
  - Access of data structures
  - Maps onto texture fetch

Grid Framework

- Input, multiple arrays
  - Different resolutions
- Grid computation
  - No inter-cell dependencies
- Kernels
  - Applied to each grid element
  - High arithmetic intensity
- Output, multiple arrays
  - Must have same resolution

Cost/Performance

- Porting to GPU non-trivial
  - Significant speedup payoff
- Efficient data structures
  - In video memory, sliver
- GPU for computing cheap
  - None of display, texture filtering
- Multi-GPU
  - Load partition, shared resources



3x3 Grid

7 active elements

2 sliver elements

Limitations

- No double precision float
  - Single float good enough
- No pixel scatter
  - Self-modifying location expensive
- Dedicated branch units
  - Both sides executed
- Grid outputs limited
  - 4/8 in DirectX 9/10, respectively

Outline

- GPU, processor array
- Compute abstraction
- Physics simulation
- Mesh video mapping
- Results, summary

Graphics API

- Complex, non-intuitive
  - for general computation
- Quest for RISC API
  - Reduced interface set core
- Full screen quad
- RISC API for rendering
  - Ray tracing

Goals

- Hide graphics API
  - Underlying DirectX 9 & 10
- Evolving hardware seamless
- Scalable
  - Multi GPU support
- Automatic multi passing
  - Overcome GPU limitations

Compute API

- Opaque device pointer(s)
  - Single, multi-GPU
- Interface objects

Abstract Object	GPU Object
Resource	Texture (2D/3D)
Kernel	Shader
Scratch	Target (2D/3D)

- Object actions
  - Create, assign, remove

Compute API (cnt'd)

- Data types
  - Scalars, vectors, matrices
- Load, store
  - Resource, scratch, respectively
- Kernel parameters
  - One time compile overhead
- Compute invocation
  - Across grid cells
- Debug, scratch dump

Format

- A single format
  - For both resource, scratch
- Four float components
- Three component vector
  - Alpha channel for control
- Three/four squared matrix
  - Three dimensional array

X	Y	Z	C

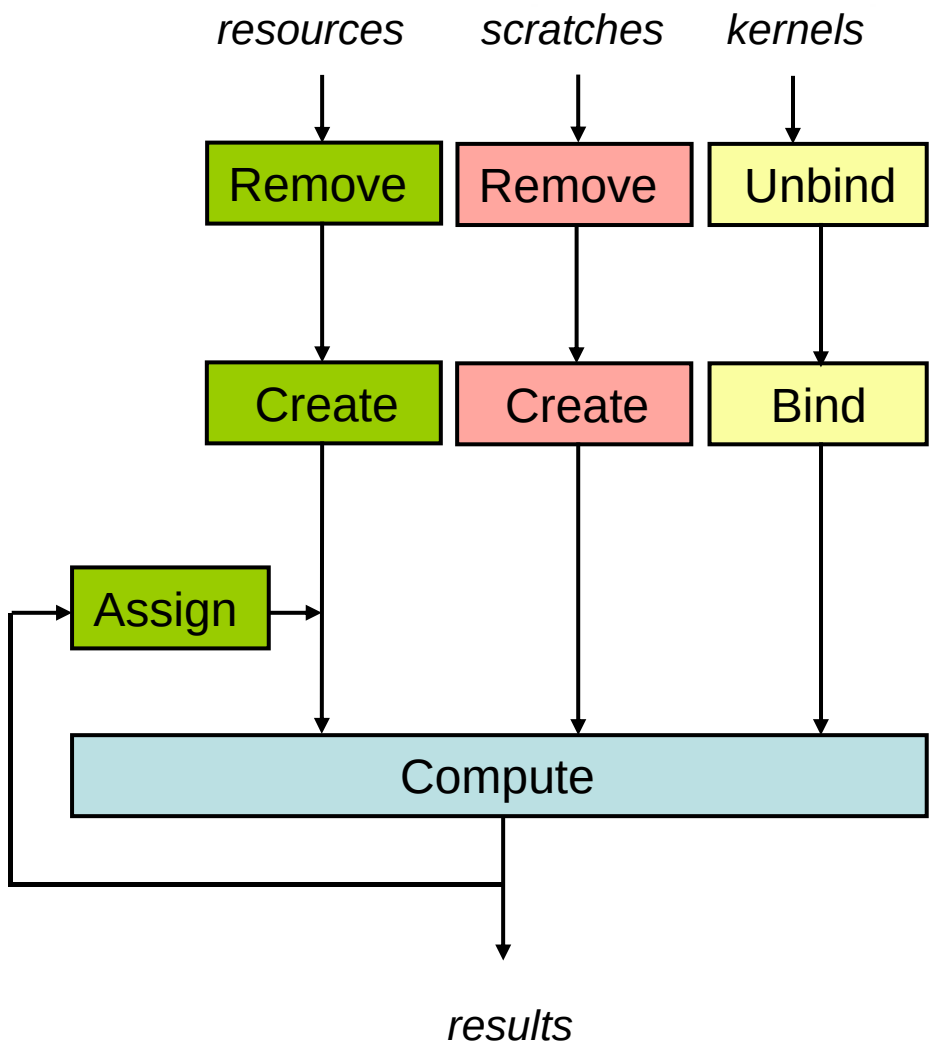
vector

slice #2	20	21	22	
slice #1	10	11	12	
slice #0	00	01	02	

matrix

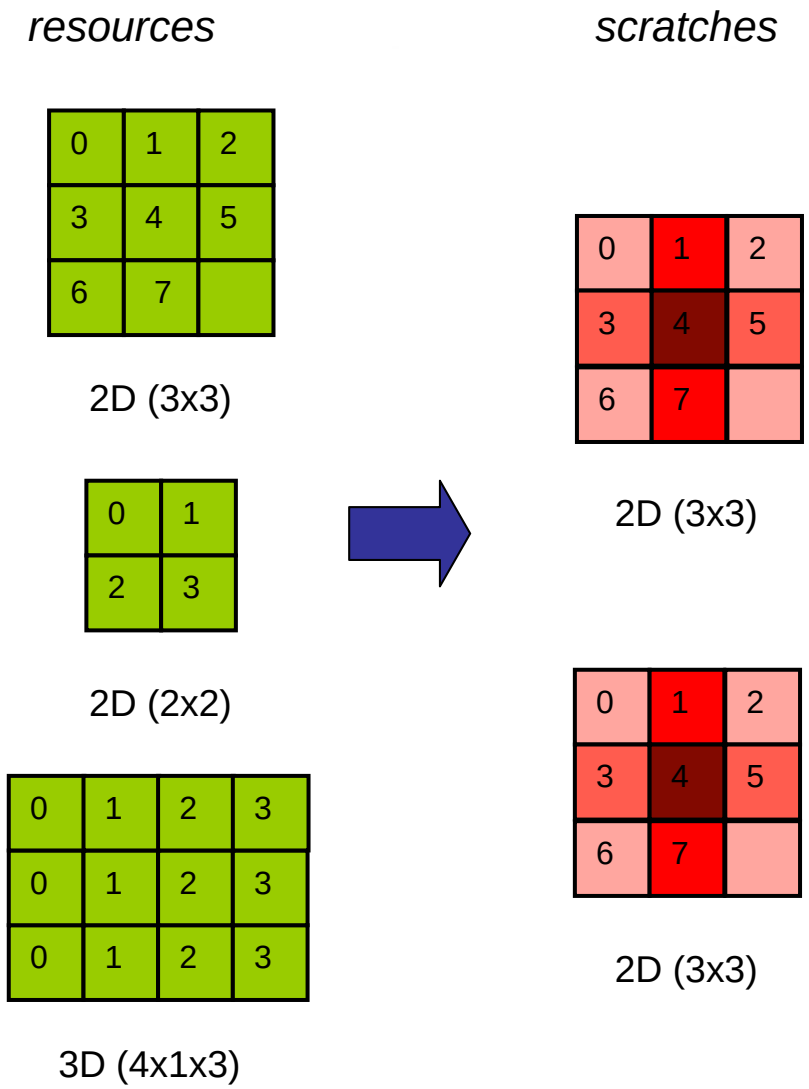
Workflow

- Iterative
  - Recirculate scratches
- Parallel setup paths
  - Resources, kernels
- Implicit correlation
  - Scratches, kernel
- Multi pass compute
  - Single kernel execution
- Results CPU access



Quad Draw

- Grid parallel traversal
  - Tiled pattern, ownership
- Point sampled grid cells
  - *VPOS* for cell index
- Relative resource address
  - Multi resolution resources
- Dependent data access
- Multi pass reduction

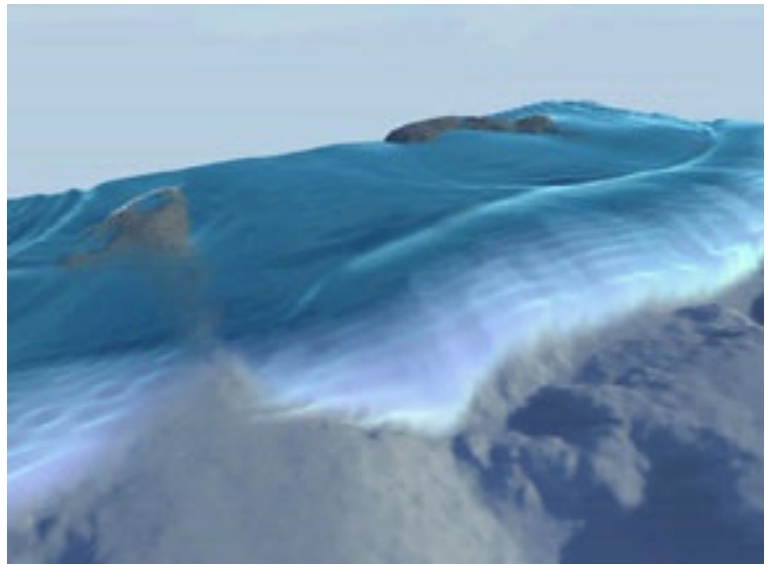


Multi GPU

- Compute bound process
- Asymmetric processing
- Computation distribution
  - Simple grid subdivision
- Synchronization overhead
  - PCI Express contention
- Shared resources

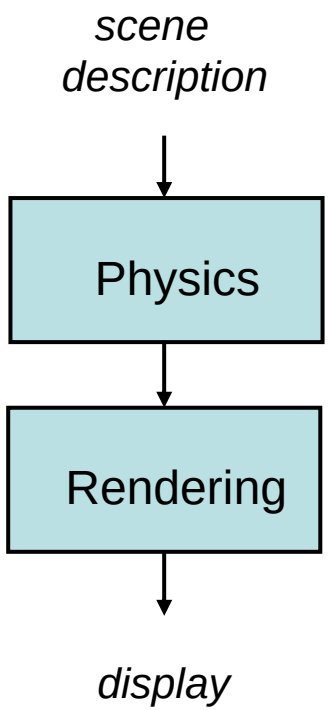
Outline

- GPU, processor array
- Compute abstraction
- Physics simulation
- Mesh video mapping
- Results, summary



Background

- Game engine
  - Simulation, visual rendering
- Physics scope
  - Rigid, deformable bodies
- Physics platforms
  - Multi core CPU, PPU, GPU
- GPU assisted physics
  - Game play, effects



FYSI

- Physics scene description format
  - *FYSL*, GPU oriented
- Consistent I/O abstraction
  - Simulation input and results
- Stream API
  - Asynchronous, discrete simulation
- GPU, CPU implementation
  - Performance analysis

FYSI (cnt'd)

- Iterative *physics pipeline*
  - *System Setup, Solver, Collision*
- Physics to rendering interface
  - Position, transform update
- Compute Abstraction Layer
  - On top of DirectX API

Programming

```
// instantiate Fysi, attach a Fysi done callback
IFysi* m_ifysi = new IFysi();
FysiUpdate* m_update = new FysiUpdate();
m_ifysi->attach(m_update);

// initialize methods
m_ifysi->init(IFysi::File, IFysi::DX9, IFysi::File);
m_ifysi->setProcessor(IFysi::GPU);
m_ifysi->setFlag(IFysi::LastStep, true);


m_ifysi->setDevice(pDevice);

// setup
std::string description = "rigid.xml";
m_ifysi->scene(description.c_str());

// simulate (fork as working thread)
Int numSteps = 10;
m_ifysi->simulate(numSteps);

// working thread loop/sleep
while(!updated) ...;

if(!m_update->getStatus()) std::cerr << m_fysi->getError();
else {
    for(int i = 0; i < numSteps; i++) {
        std::cout << m_fysi->getResults(i);
    }
}
```

FYSL

- Simulation control
  - Type, time step
- Actors, hierarchical
  - Bounding volumes, meshes
  - Linear, angular motion
- Joints, motion constraints
  - Spring, distance
- Feedback
  - Path decision

Sample

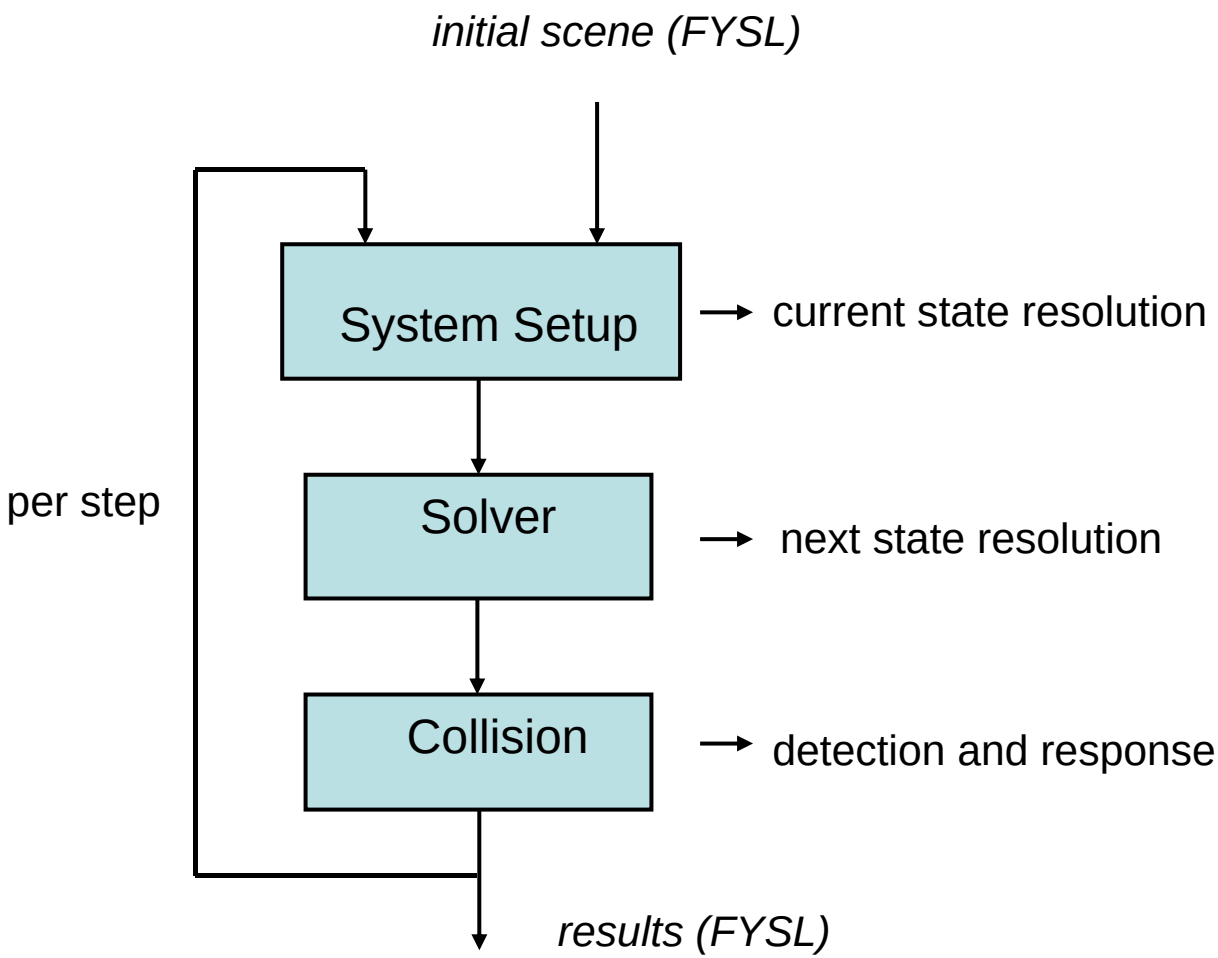
```
<Actor>
     <Name>a0</Name>
     <Type>static</Type>
     <Id>0</Id>
     <Shape>
          <Box>
               <Position>-1. .0 -1.</Position>
               <Radius>2. .5 -1.5</Radius>
          </Box>
     </Shape>
     <Dynamics>
          <Linear>
               <Mass>10</Mass>
               <Velocity>10. .0 .0</Velocity>
               <Force>.5 1.2 -.5</Force>
          </Linear>
     </Dynamics>
</Actor>
```

Actor definition

```
<Joints>
     <Joint>
          <Distance>
               <ActorPair>0 1</ActorPair>
               <Min>.1</Min>
               <Max>1.</Max>
          </Distance>
     </Joint>
     <Joint>
          <Spring>
               <ActorPair>1 2</ActorPair>
               <RestLength>2.3</RestLength>
               <Stiffness>1.5</Stiffness>
               <Damping>3.2</Damping>
          </Spring>
     </Joint>
</Joints>
```

Joints definition

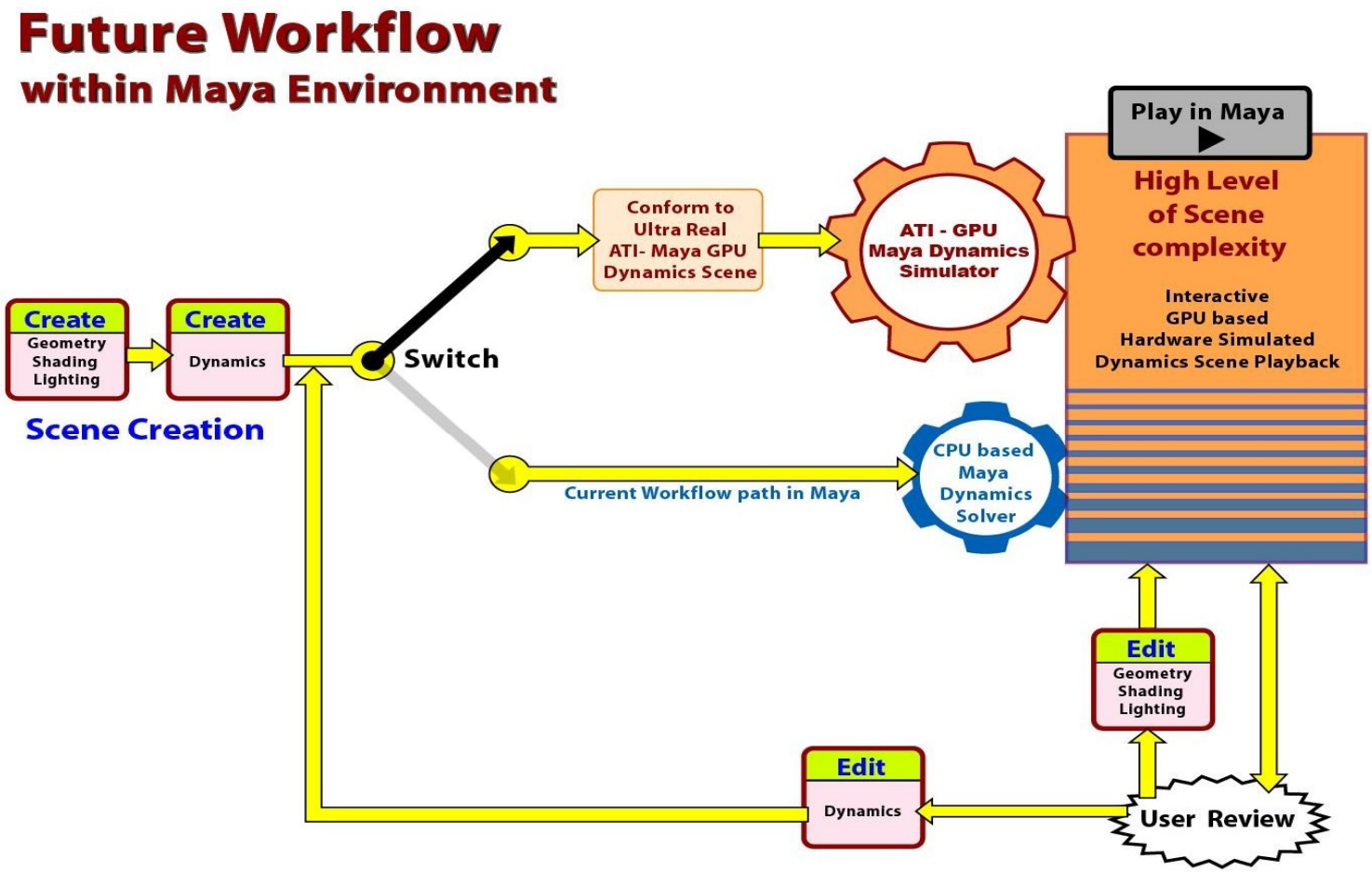
Physics Pipeline



Simulation

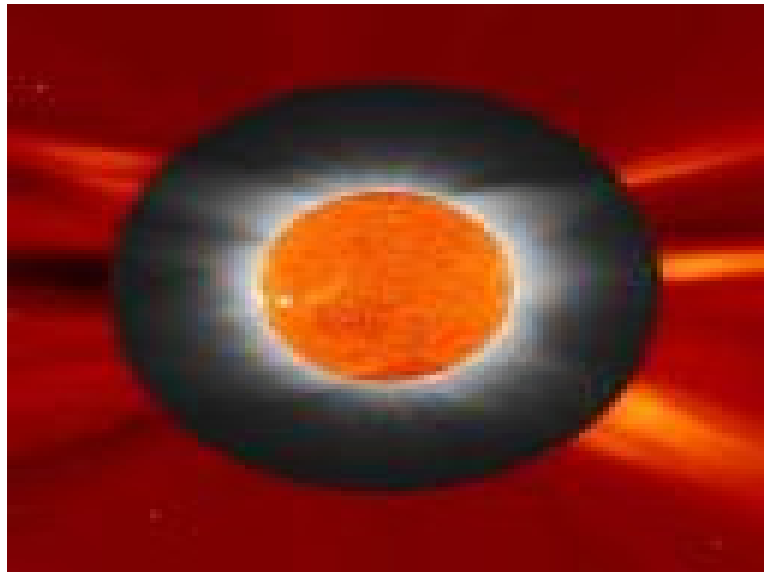
- Grid based simulation
  - Euler method (x(t + dt) = x(t) + t * dx/dt;)
- Geometry, property resources
  - Texture array
- Portable shading library
- Resume from last step
  - Result caching
- Adaptive time step

GPU Dynamics



Outline

- GPU, processor array
- Compute abstraction
- Physics simulation
- Mesh video mapping
- Results, summary

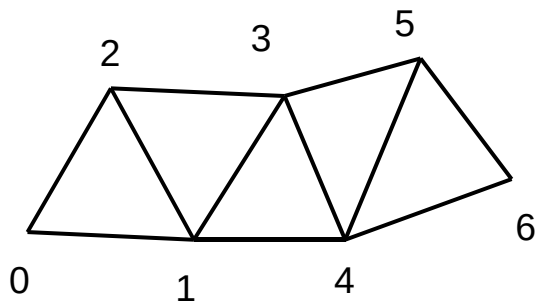


Geometry

- Object space representation
  - Precise, CPU equivalent
- Geometry in video memory
  - Multiple arbitrary meshes
- GPU texture addressability
  - 4K by 4K for 2D
- Mesh representation
  - 1D vertex and index buffers

Vertex Mesh

- Vertex and index buffers
  - Positions and faces, respectively
- Counter clockwise triangles



triangular mesh

position # 0
position # 1
position # 2
position # 3
position # 4
position # 5
position # 6

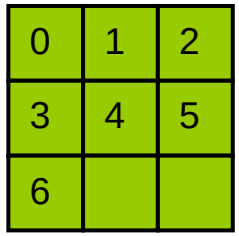
vertex buffer

face # 0 (0, 1, 2)
face # 1 (2, 1, 3)
face # 2 (1, 4, 3)
face # 3 (3, 4, 5)
face # 4 (5, 4, 6)

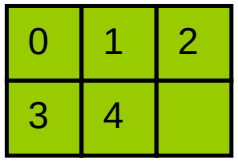
index buffer

Texture Mesh

- Mesh as a resource
  - 2D video memory array
- Three floats element
  - {x,y,z} and {i0,i1,i2}
- Arbitrary dimensions
  - Padded as necessary
- Dependent texture
  - For each triangle access



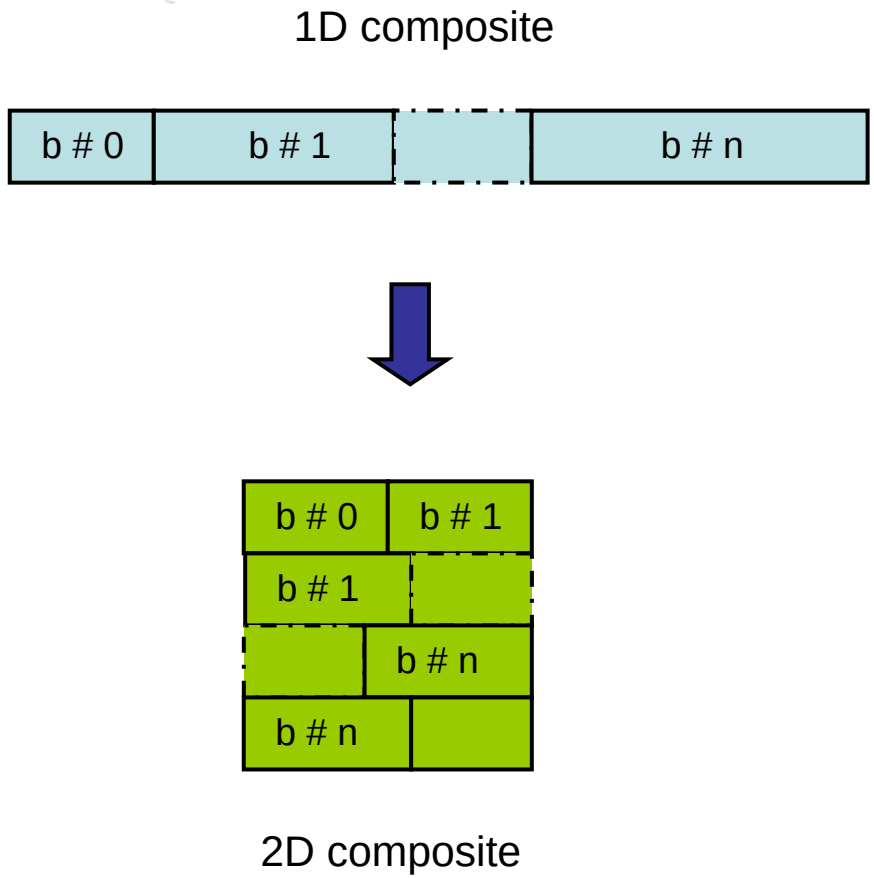
vertex buffer (3x3)



index buffer (3x2)

Compositing

- Many mesh scene
- 16 textures constraint
- A single super mesh
  - Coalesces multiple meshes
  - Vertex and index uber buffer
- Index buffer offset
- Boundary sub mesh id

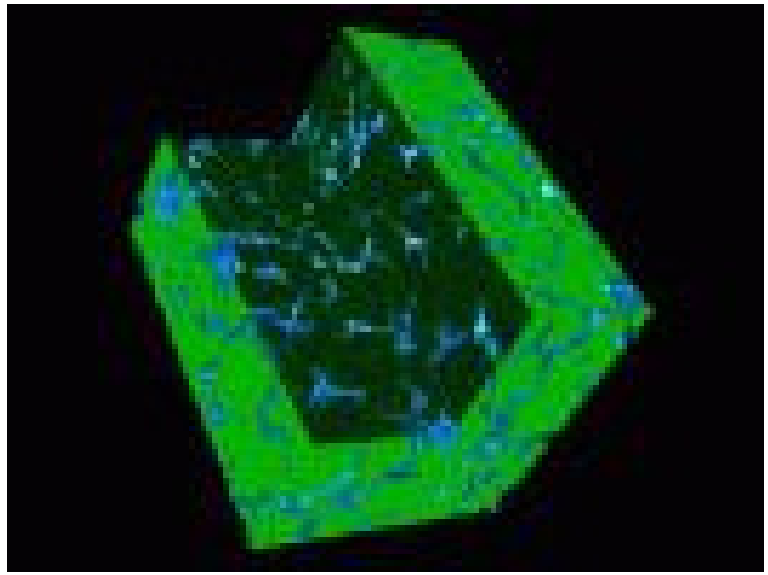


Traversal

- Low creation overhead
- Sub mesh collision
  - Tri-tri intersection, contact
- Avoids self intersection
- Multi pass compute
  - Intersect, response, integrate
- Dependent texture access
  - Face to vertex fetch

Outline

- GPU, processor array
- Compute abstraction
- Physics simulation
- Mesh video mapping
- Results, summary



Goals

- GPU superior to CPU
  - Collision detection, response
- Understand compute behavior
  - Grid distribution
  - Flow control, multi pass
- Scalability across GPUs
  - More pipes, ALUs

Methodology

- Single CPU
  - No dual core, no SSE2
- System performance
  - Physics process, no rendering
- Shape based benchmarks
  - Linear motion
- Absolute, normalized results

Processors

Processor	Pipes	ALUs	Core (MHz)	Memory (MHz)
Pentium 4	1	1(4)	3400	533
Radeon X1600	4	3	600	700
Radeon X1800	16	1	625	750
Radeon X1900	16	3	650	775
GeForce 7800	24	2	580	865
GeForce 7900	24	2	650	800

Arithmetic Intensity

Shader Type	Ops	ALU	Texture	Ratio
volume-volume	119	87	5	17.40
mesh-mesh	300	241	8	30.13
mesh update	67	53	7	7.57
impact	102	72	19	3.79

GPU Scalability

Benchmark	CPU	X1600	X1800	X1900	7800	7900
aabb256x256	101404 *(0.0004)*	47 *(1.000)*	31 *(1.516)*	15 *(3.133)*	203 *(0.231)*	220 *(0.213)*
sphere256x256	74793 *(0.0004)*	31 *(1.000)*	15 *(2.067)*	15 *(2.067)*	188 *(0.164)*	201 *(0.154)*
mesh64x64 *(62471 tris)*	197508 *(0.0001)*	32 *(1.000)*	16 *(2.000)*	30 *(1.066)*	187 *(0.171)*	247 *(0.129)*
tetrahedron128x128	562768 *(0.0002)*	125 *(1.000)*	110 *(1.136)*	47 *(2.659)*	703 *(0.177)*	482 *(0.259)*

Compute draw calls (figures in msec)

Observations

- CPU wins for small grids
  - Expected, setup overhead
- GPU faster elsewhere
  - Up to an order of magnitude
- GPU Scalability
  - Flow control might be stalling
- GeForce 7800/7900 slower
  - Across benchmarks

Futures

- Unified shader architecture
- Higher concurrency level
  - 64 pixel engines
  - Flexible mix of scalar/vector
- DirectX 10
  - Constant buffers
  - Texture array, indexing
  - Non-power-of-2 3D textures
  - 3D render target

Summary

- GPU game computing
  - Still a challenge
- Parallel programming
  - Macro and micro level
- Software productivity
  - Tools, tools, tools…
- Emerging CPU/GPU platforms
  - Adaptive load balance

Thank You!

Questions?