GPGPU Computing Applications in Graphics and Game Development

CPE613 Topical Seminar

7/1/14

By Falco Girgis

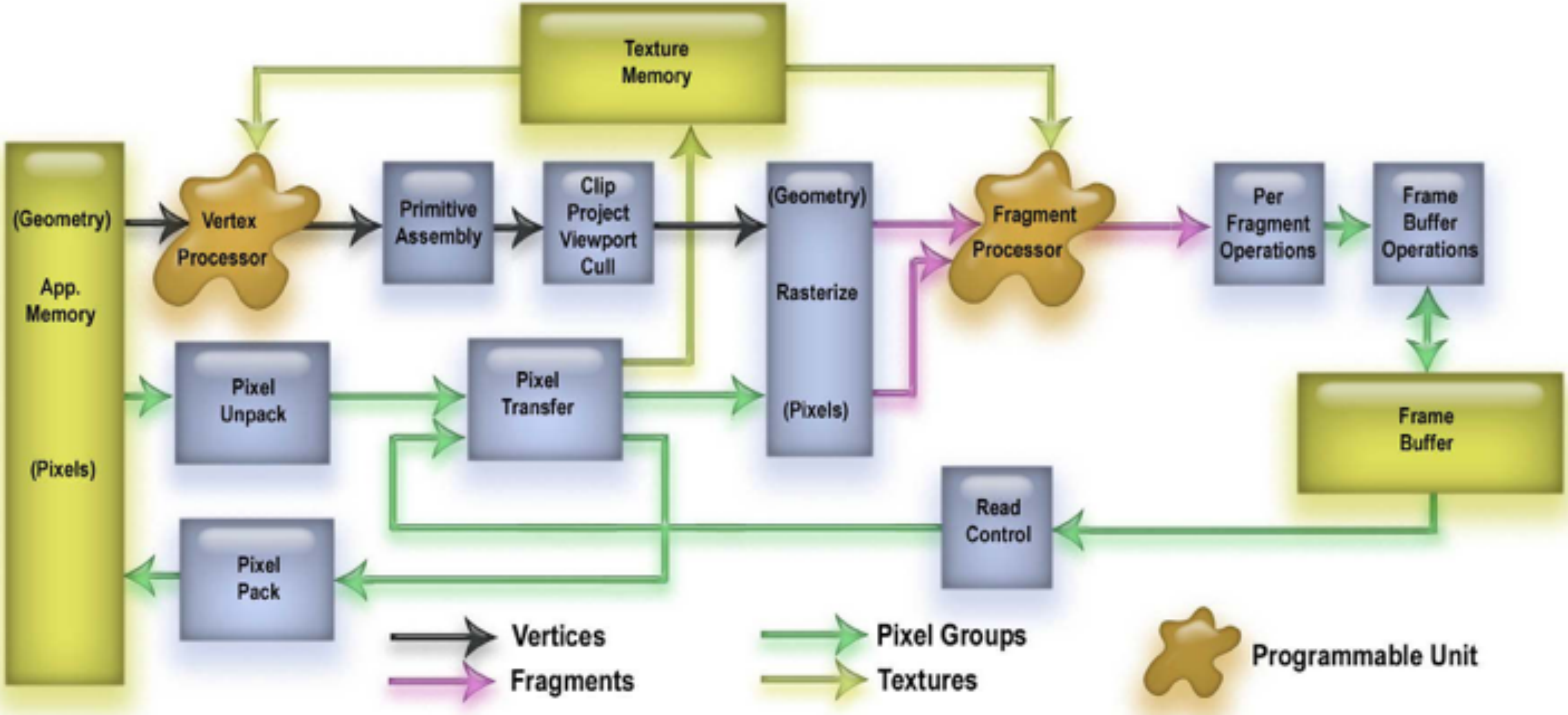
Outline

- Motivation
- Intro to Graphics Pipeline
- Intro to Shaders
- OpenCL/OpenGL Interop Model
- GPGPU with Game Development
- Collision Detection
- Physics Engines
- Artificial Intelligence
- Shortcomings
- Future of Interop

Motivation

- Highly parallelized GPU architecture
  - significantly more FLOPS
- Ubiquity of programmable GPUs
  - desktops, cell phones, game consoles
- Memory Wall - Processor vs RAM speeds
  - GPUs have greater memory bandwidth
- Increasing GPU speeds - cubed Moore’s Law
  - due to data-parallel streaming

GPU Graphics Pipeline



Introduction to Shaders

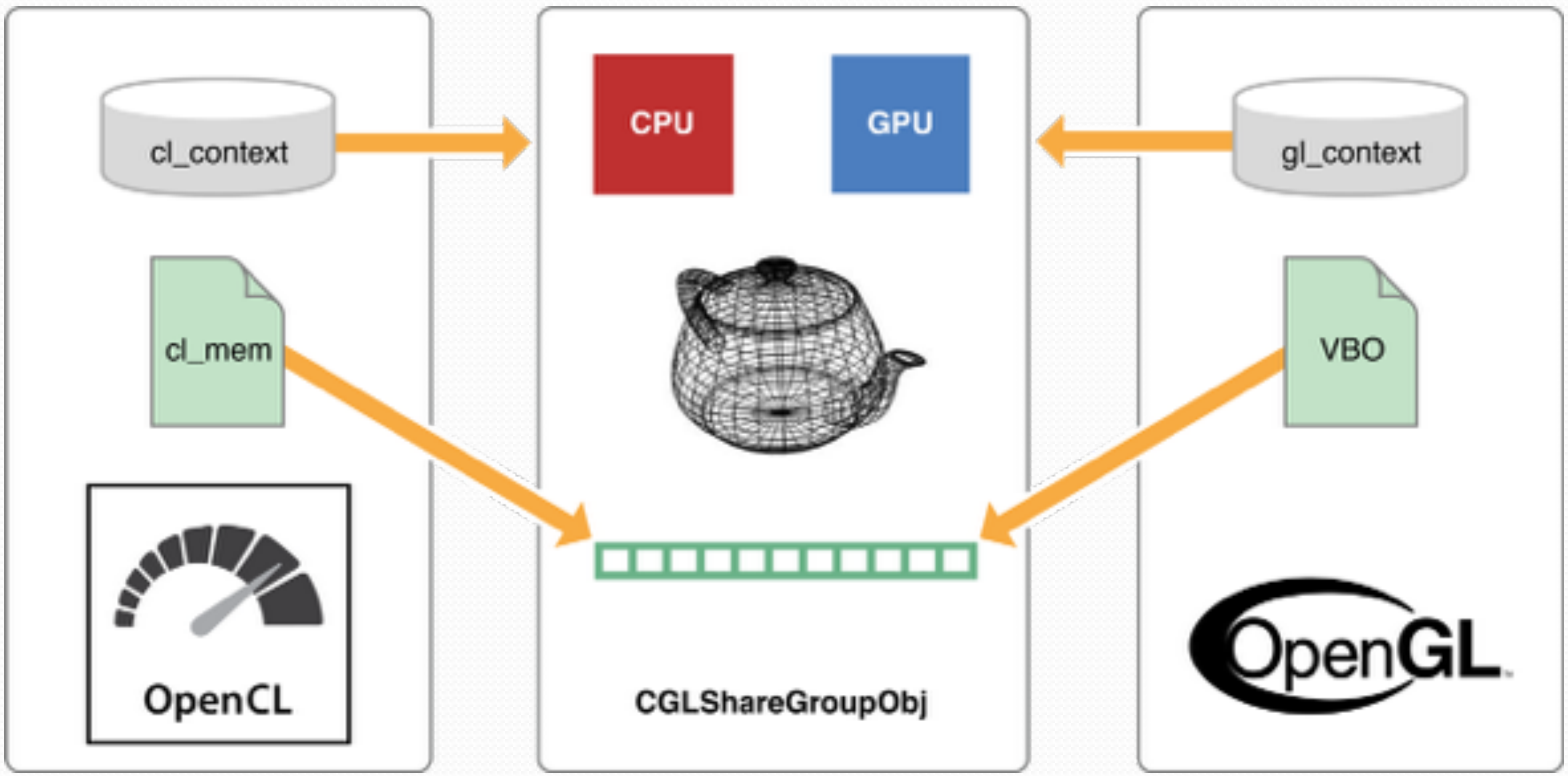
- Programmable portions of graphics pipeline
- Analogous to GPGPU Kernels
- Data-level Parallel
  - hundreds of threads operating in parallel
- Vertex Shaders
  - thread scheduled for every vertex submission
- Fragment Shaders
  - thread scheduled for every filled pixel/texel

Example GLSL Vertex Shader

```
varying vec4 diffuseColor;
varying vec3 fragNormal;
varying vec3 lightVector;

uniform vec3 eyeSpaceLightVector;


//Executed for every vertex being rendered
//It’s like an OpenCL/CUDA Kernel
void main(void) {
    vec3 eyeSpaceVertex = vec3(gl_ModelViewMatrix * gl_Vertex);
    lightVector= vec3(normalize (eyeSpaceLightVector - eyeSpaceVertex));
    fragNormal = normalize(gl_NormalMatrix * gl_Normal);

    diffuseColor = gl_Color;
    gl_Position = gl_ModelViewProjectionMatrix * glVertex;
}
```

OpenCL and OpenGL Interop

- OpenGL - Vertex Buffer Object (VBO)
  - device buffer storing geometry data used to render a model
- OpenCL - cl_mem
  - generic device buffer handle



Sharing Data between CL and GL

- Basic Approach
  1. Initialize CL context from a GL ShareGroupObj
  2. Create objects in GL as usual
  3. Create references to GL objects in CL
  4. Synchronize and Swap Ownership between two
  5. Release CL reference then destroy GL object

- Examples
  1. Map GL texture or render-buffer to CL image
     - Perform image processing in OpenCL, render with OpenGL
     - clCreateFromGLTexture(context, flags, target, miplevel, texture, error)
  2. Map GL VBO to cl_mem buffer
     - CL updates geometry, colors, or normals. GL renders.
     - gcl_gl_create_ptr_from_buffer(id)

General GL/CL Interop Considerations

- Shared buffer approach avoids copying between GL and CL contexts
- Synchronization between GL, CL, and Host
  - CL/Host - CL command queue
  - GL/Host - glFlush()
- Keep as much logic as possible on the GPU
  - pipeline data-parallel tasks on GPU
  - minimizes transfer overhead requirements
  - maximizes GPU utilization

Game Development Applications

- Texture Processing
  - advanced pre or post processing applied to GL surfaces outside the graphics pipeline
- Collision Detection
  - Broad-phase
  - Contact creation
- Particles and Physics Engines
  - Simulating Rigid Bodies and interactions
- Artificial Intelligence

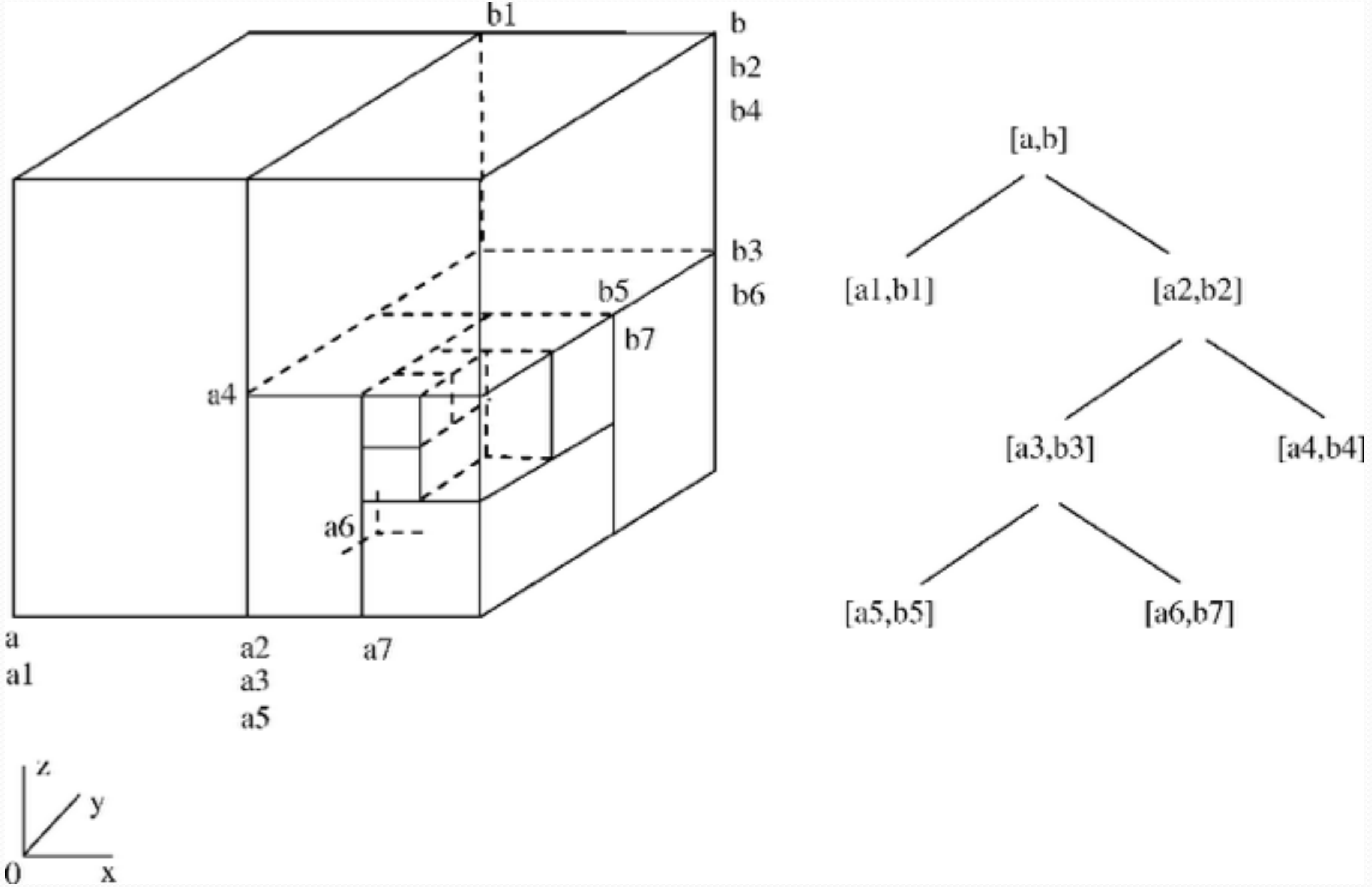
Texture Postprocessing

- Radial blur applied to whole frame buffer

Broad Phase Collision Detection

- Spatial Partitioning
  - subset of n-body problem
  - partition scene into smaller segments
  - create graph grouping nearby objects
  - reduces time-complexity of actual collision checks
    - not checking everything against everything
- Objects can be processed independently
  - highly data-level parallel
  - kernel thread per object

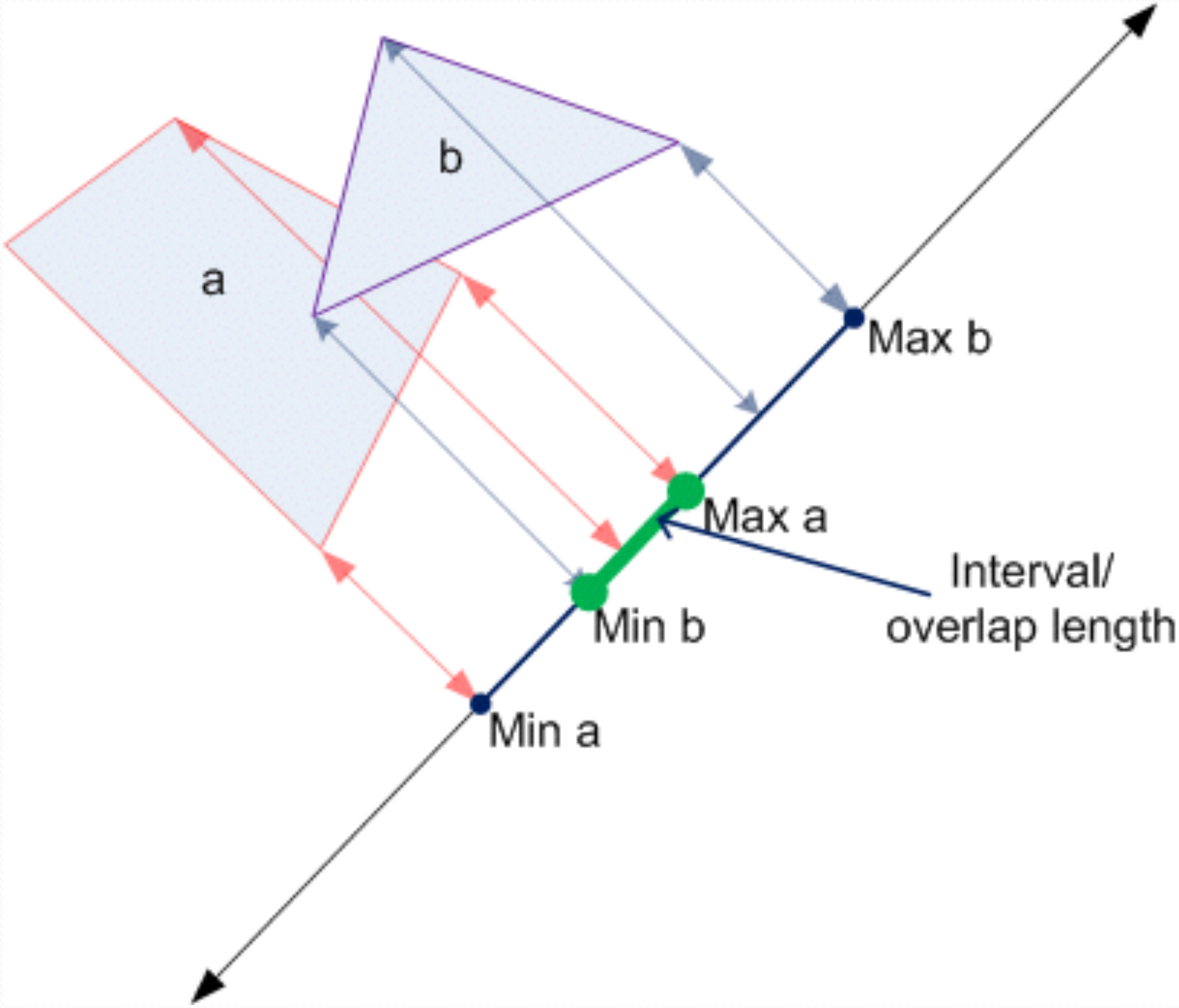
Binary Space Partitioning



Contact Generation

- "Fine" Collision Detection
  - Very computationally intensive
    - complex geometry, points of contact, depth, normals, lost of linear algebra
  - Each potentially colliding pair is independent
    - CL kernel checks collision against two bodies
    - one thread per collision check
    - also highly parallelizable

Separating Axis Convex Polygon Collision



258 Separating Axis Contacts

Physics Simulation Pipeline

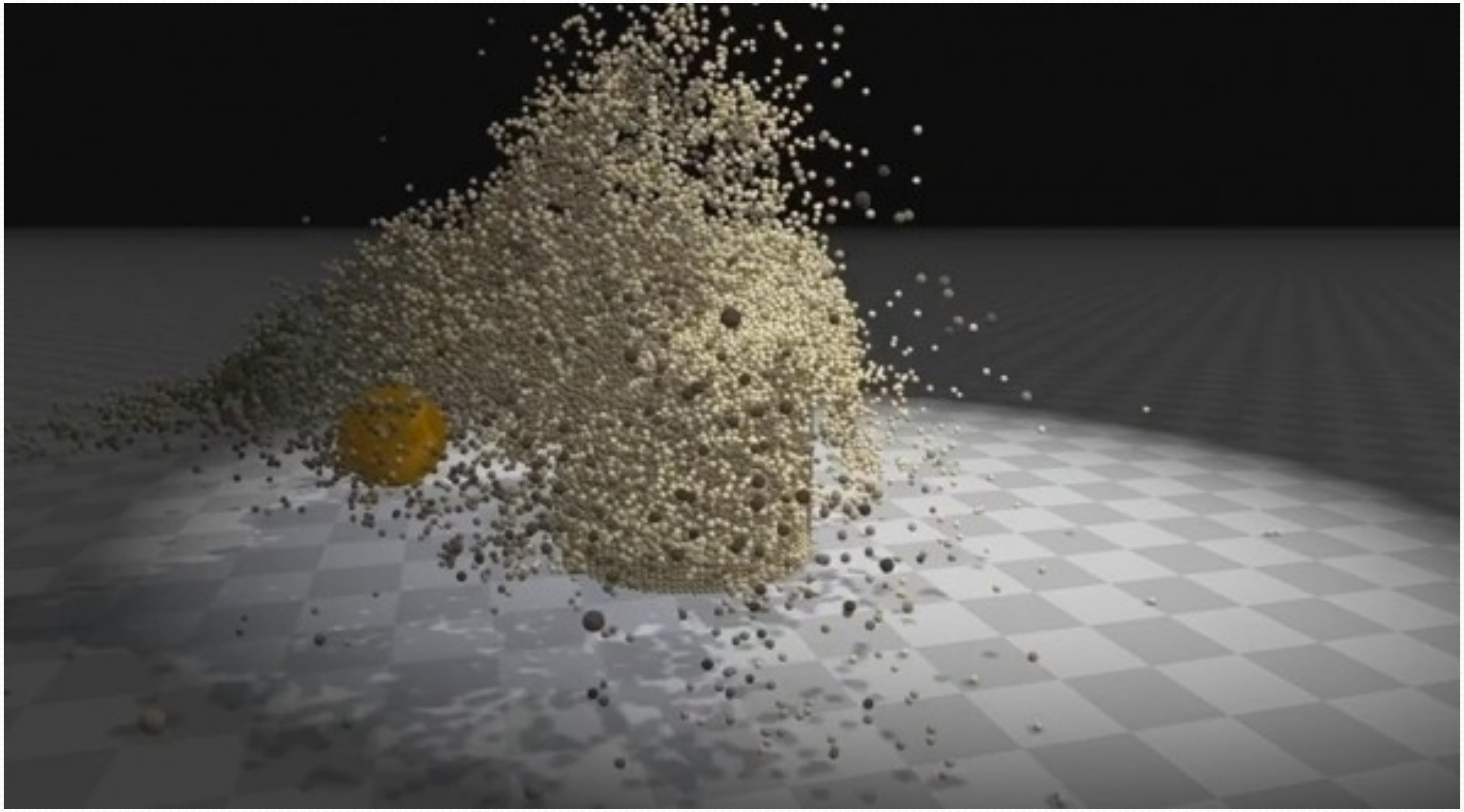
1. Force Application
    - accumulate outside force influences
2. Integration
    - update acceleration, velocity, and position
3. Collision Detection
    - spatial partitioning
    - contact generation
4. Contact Resolution
    - penetration resolution
    - velocity resolution

Physics Pipeline Considerations

- Applies same operations to large groups of independent objects
  - highly data-level parallel
- Broken into discrete pipeline stages
  - each stage can be handled by a set of kernels
  - entire pipeline can execute on the GPU without requiring additional data
    - significantly reduces transfer overhead

NVidia PhysX Engine

- Uses CUDA and DirectX

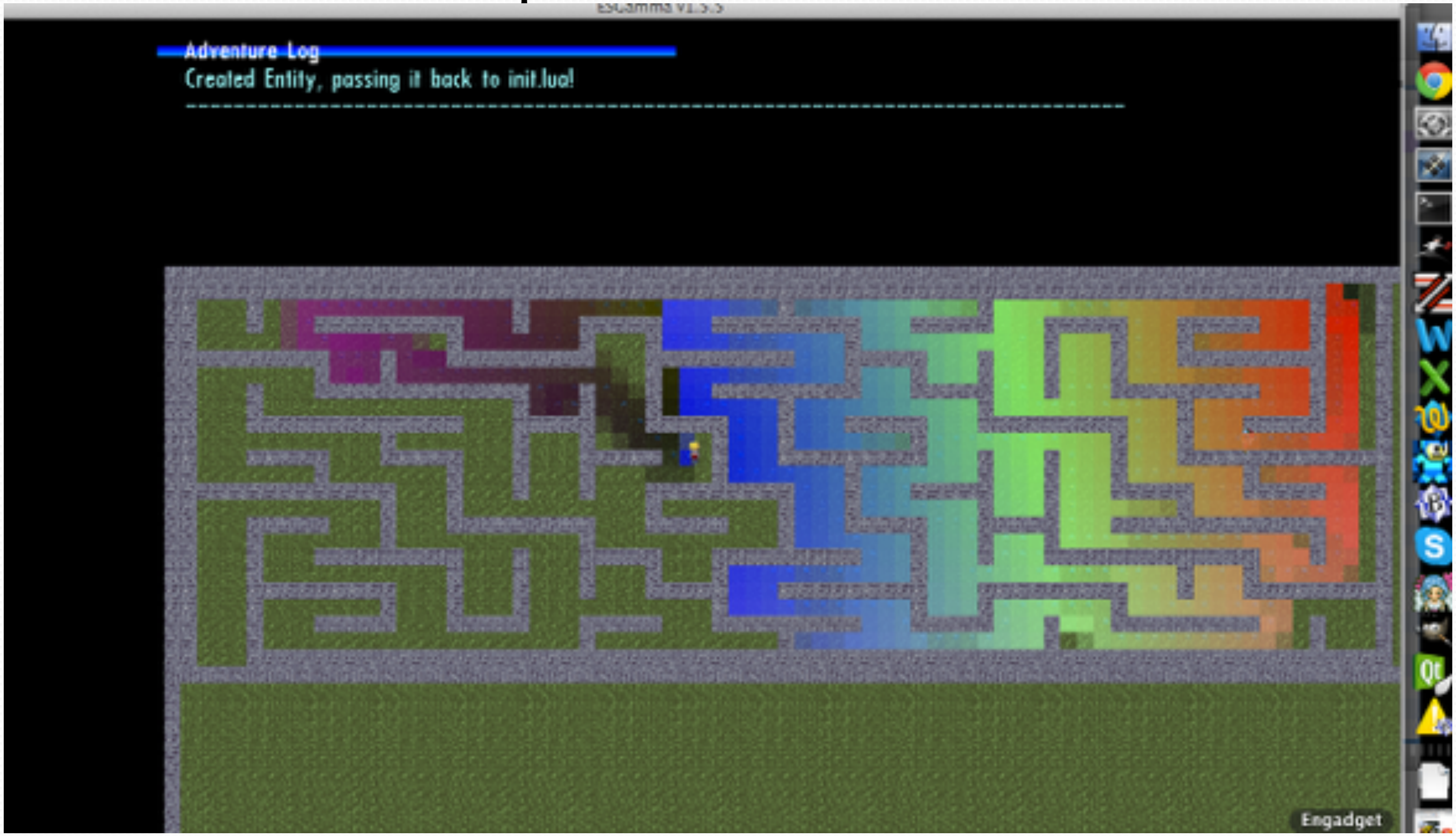


Artificial Intelligence

- Traditionally CPU-bound problems
- Machine-Based Learning
  - training phases require massive amounts of data processing
- Path-Finding
  - Many algorithms must exhaustively search potential paths
  - GPU can process many paths in parallel
- Genetic Algorithms
  - determine fitness value per individual
    - population is a shared data structure
    - kernel calculates fitness of each individual in parallel

A* Pathfinding Algorithm

- least-cost path requires many graph traversals
  - can be done in parallel on the GPU



Shortcomings

- CPU requiring intermediate data from GPU pipeline
  - requires a transfer back to the host
  - may require a second transfer back to GPU
- Branch-intensive game logic
  - significantly reduces GPU performance
  - causes warp divergence
- Inefficient for small sets of data
  - transfer overhead is greater than computation time
- OpenGL and OpenCL interop is still relatively new
  - not every driver implementation supports interop

GPGPU with Game Development in the Future

- CPU and GPU memory unification trend
  - will significantly reduce transaction overhead
  - allows less rigid GPU pipelines, since CPU can access intermediate data more quickly
- GLSL “Compute” Shaders
  - analogous to OpenCL/CUDA kernels
  - general-purpose processing in graphics APIs
- GPGPU Advances
  - dynamic parallelism
  - shared virtual memory
  - pipes

Resources

1. https://developer.apple.com/library/mac/documentation/Performance/Conceptual/OpenCL_MacProgGuide/shareGroups/shareGroups.html
2. https://developer.apple.com/library/mac/documentation/Performance/Conceptual/OpenCL_MacProgGuide/SynchronizingCLandGL/SynchronizingCLandGL.html#//apple_ref/doc/uid/TP40008312-CH18-SW1
3. https://software.intel.com/en-us/articles/opencl-and-opengl-interoperability-tutorial
4. http://sa10.idav.ucdavis.edu/docs/sa10-dg-opencl-gl-interop.pdf
5. https://developer.nvidia.com/gpu-ai-path-finding
6. http://www.geforce.com/hardware/technology/physx
7. http://what-when-how.com/artificial-intelligence/ia-algorithm-acceleration-using-gpus-artificial-intelligence/
8. http://www.opengl.org/wiki/Compute_Shader
9. https://www.khronos.org/news/press/khronos-releases-opencl-2.0