

# OpenCL Game Physics

Bullet: A Case Study in Optimizing Physics Middleware for the GPU

Erwin Coumans

# Overview

- Introduction
- Particle Physics Pipeline from the NVIDIA SDK
  - Uniform grid, radix or bitonic sort, prefix scan, Jacobi
- Rigid Body Physics Pipeline
  - Parallel Neighbor search using dynamic BVH trees
  - Neighboring Pair Management
  - Convex Collision Detection: GJK in OpenCL on GPU
  - Concave Collision Detection using BVHs
  - Parallel Constraint Solving using PGS
- OpenCL cross-platform and debugging

# Introduction

- Bullet is an open source Physics SDK used by game developers and movie studios
- PC, Mac, iPhone, Wii, Xbox 360, PlayStation 3
- Bullet 3.x will support OpenCL acceleration
  - Simplified rigid body pipeline fully running on GPU
  - Developer can mix stages between CPU and GPU
- Implementation is available, links at the end

# Some games using Bullet Physics



# Some movies using Bullet Physics

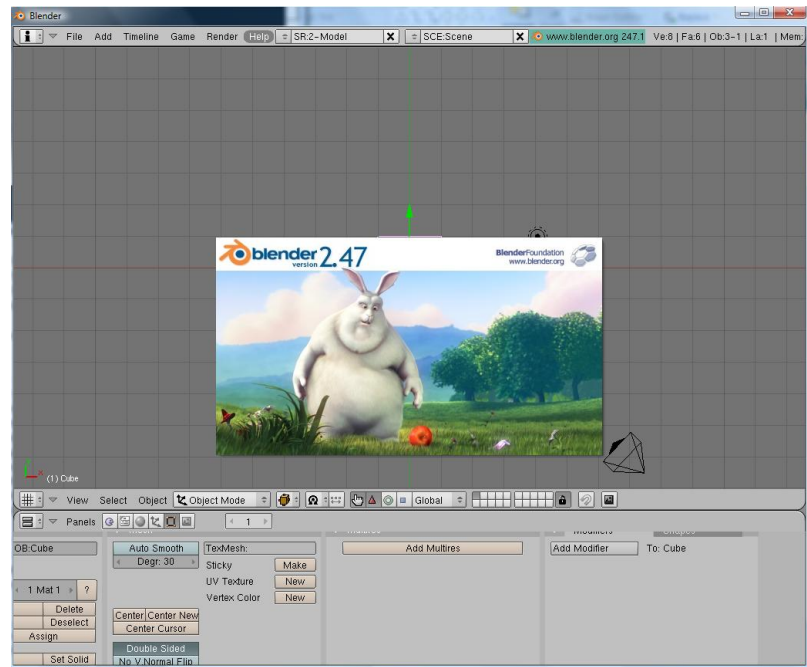


2012

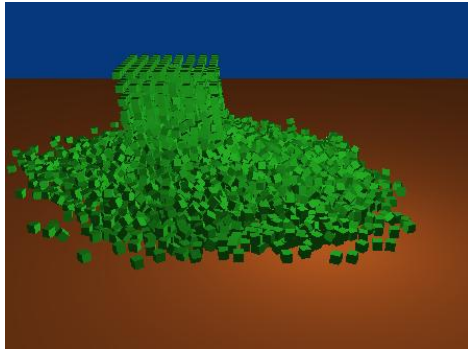
The year '2012' is displayed in large, blue, 3D, metallic letters on a dark background.

# Authoring tools

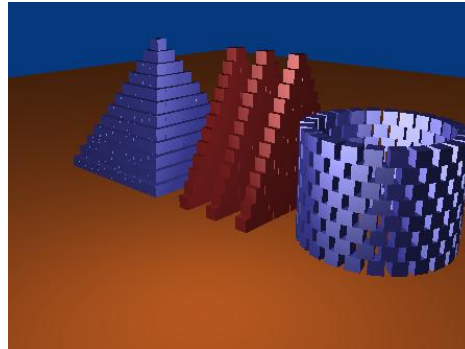
- Maya Dynamica Plugin
- Cinema 4D 11.5
- Blender



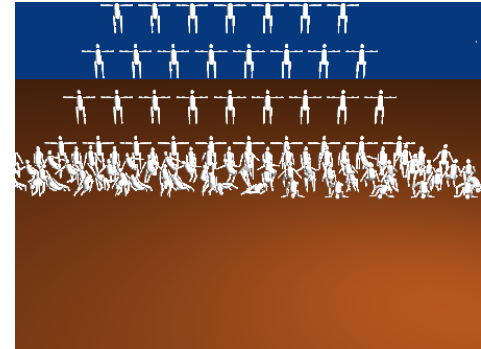
# Rigid Body Scenarios



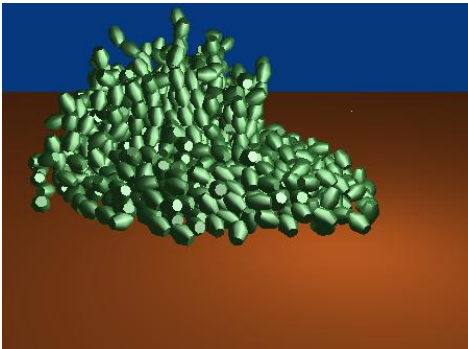
3000 falling boxes



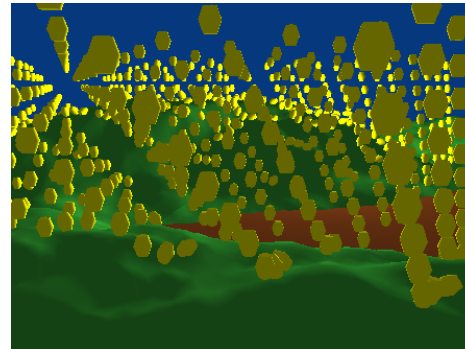
1000 stacked boxes



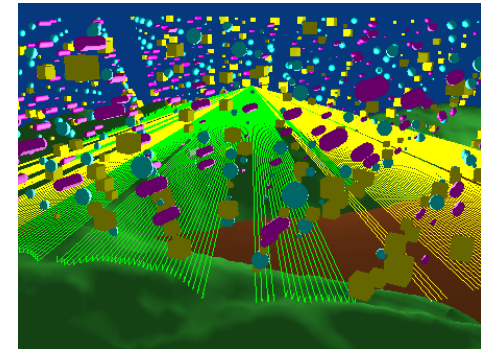
136 ragdolls



1000 convex hulls

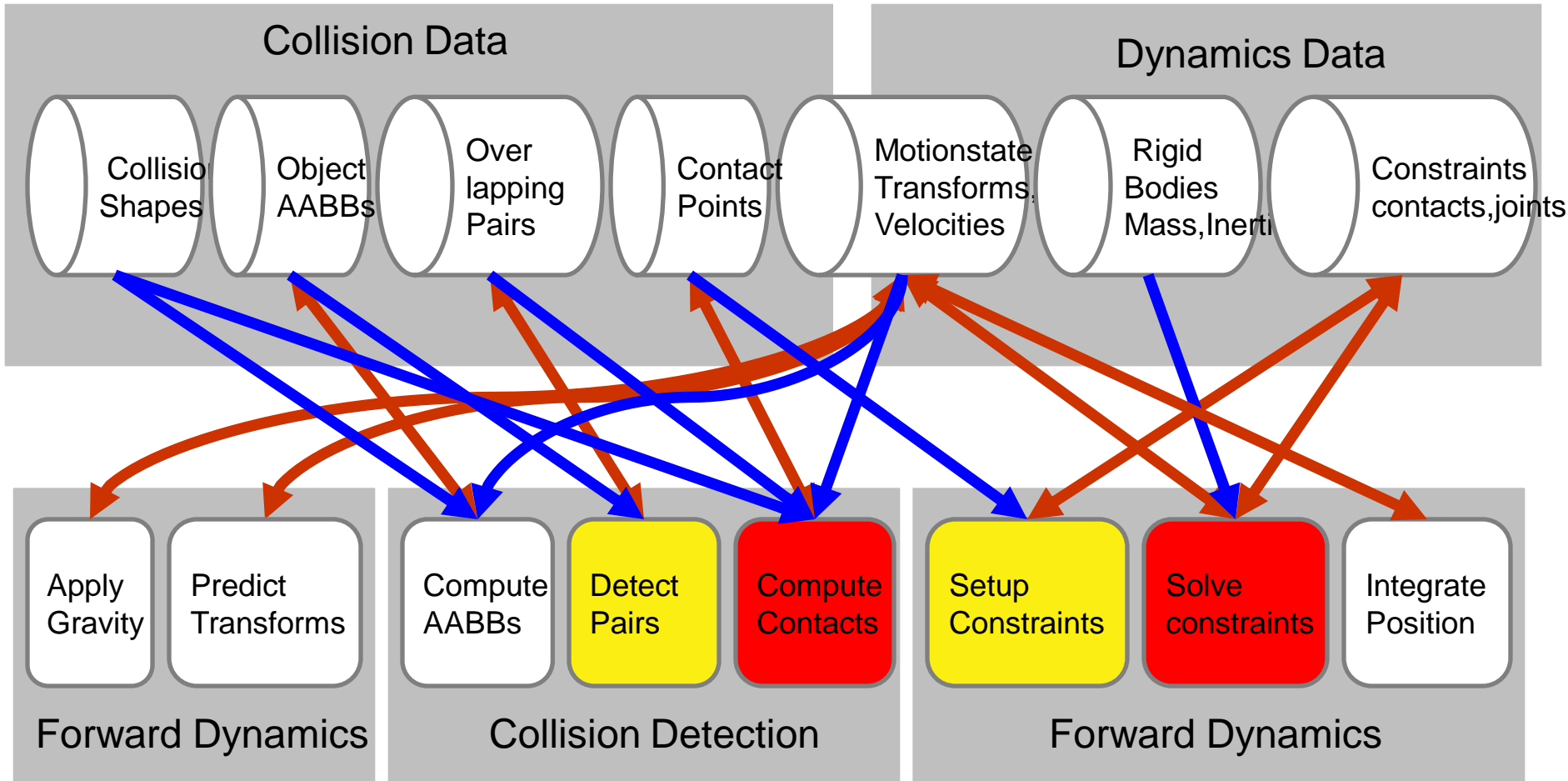


1000 convex  
against trimesh



ray casts against 1000  
primitives and  
trimesh

# Performance bottlenecks

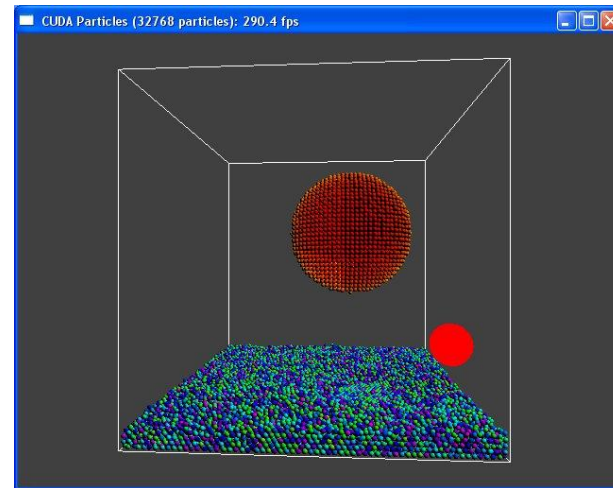
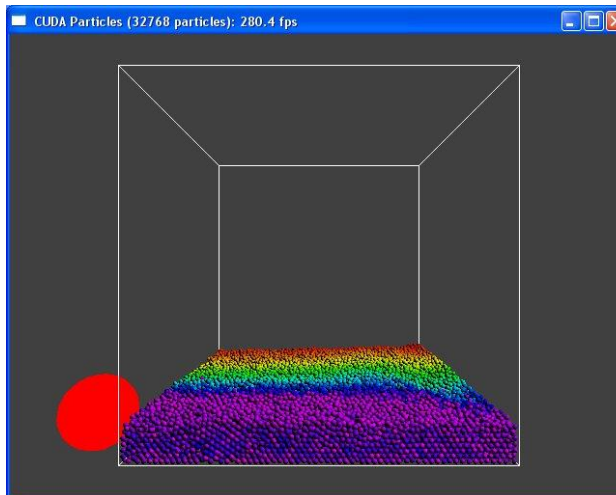
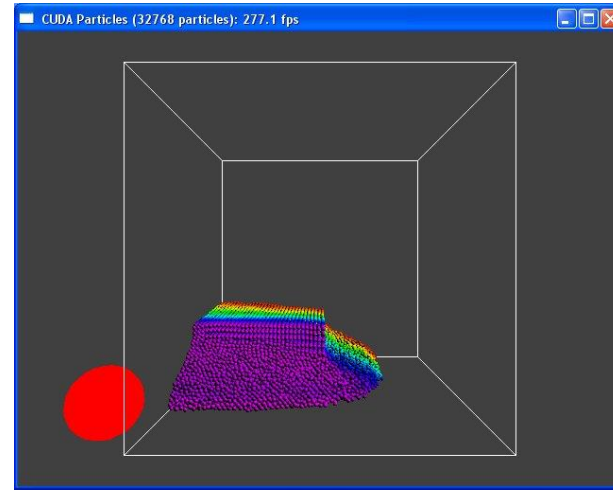
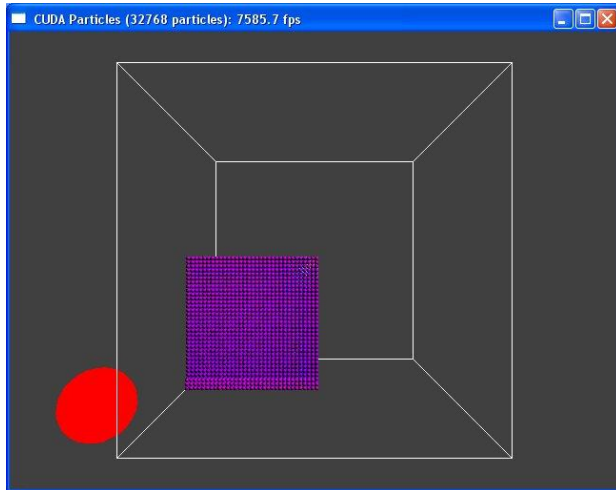




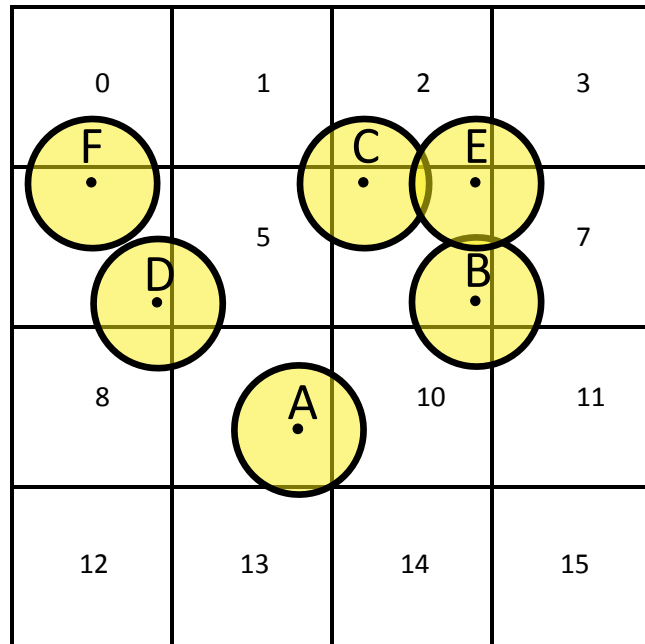
# Leveraging the NVidia SDK

- Radix sort, bitonic sort
- Prefix scan, compaction
- Examples how to use fast shared memory
- Uniform Grid example in Particle Demo

# Particle Physics CUDA and OpenCL Demo

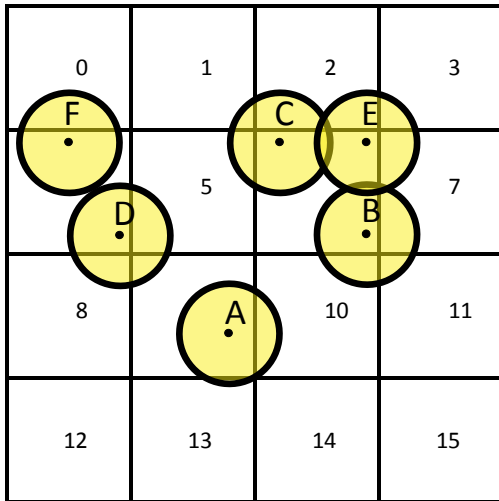


# Uniform Grid



Cell ID	Count	Particle ID
0	0	
1	0	
2	0	
3	0	
4	2	D,F
5	0	
6	3	B,C,E
7	0	
8	0	
9	1	A
10	0	
11	0	
12	0	
13	0	
14	0	
15	0	

# Sorting Particles per Cell



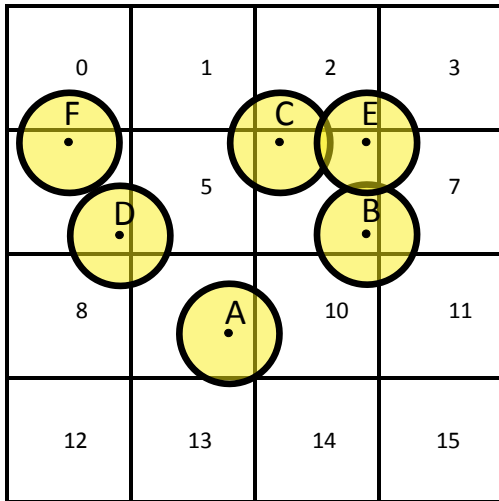
Cell Index	Cell Start
0	
1	
2	
3	
4	0
5	
6	2
7	
8	
9	5
10	
11	
12	
13	
14	
15	

Array Index	Unsorted Cell ID, Particle ID	Sorted Cell ID Particle ID
0	9, A	4,D
1	6,B	4,F
2	6,C	6,B
3	4,D	6,C
4	6,E	6,E
5	4,F	9,A

# Neighbor search

- Calculate grid index of particle center
- Parallel Radix or Bitonic Sorted Hash Array
- Search 27 neighboring cells
  - Can be reduced to 14 because of symmetry
- Interaction happens during search
  - No need to store neighbor information
- Jacobi iteration: independent interactions

# Interacting Particle Pairs



Array Index	Sorted Cell ID Particle ID
0	4,D
1	4,F
2	6,B
3	6,C
4	6,E
5	9,A

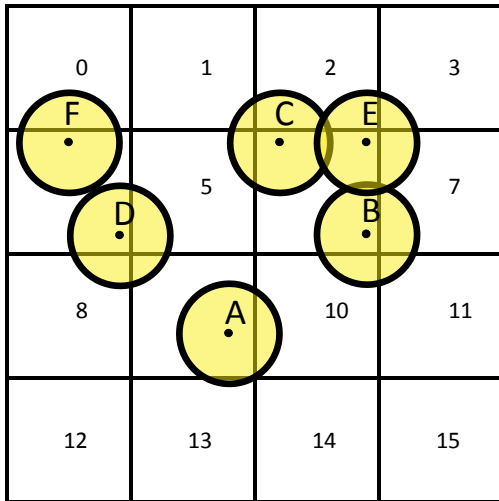
Interacting Particle Pairs
D,F
B,C
B,E
C,E
A,D
A,F
A,B
A,C
A,E

# Using the GPU Uniform Grid as part of the Bullet CPU pipeline

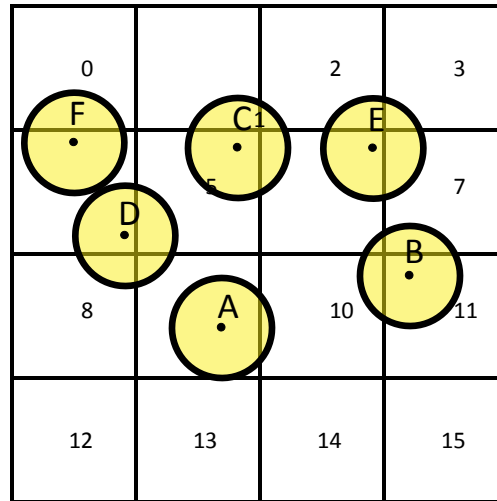
- Available through btCudaBroadphase
- Reduce bandwidth and avoid sending all pairs
- Bullet requires persistent contact pairs
  - to store cached solver information (warm-starting)
- Pre-allocate pairs for each object

# Persistent Pairs

Before



After

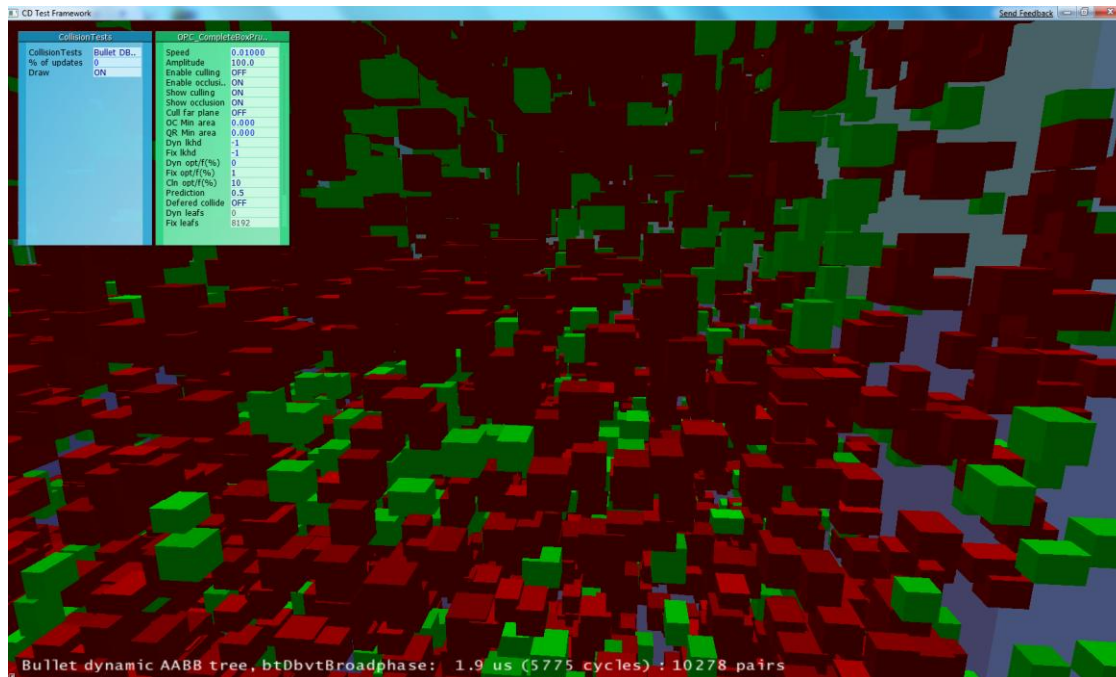


Particle Pairs Before	After	Differences
D,F	D,F	A,B removed
B,C	B,C	B,C removed
B,E	B,E	C,F added
C,E	C,E	C,D added
A,D	A,D	
A,F	A,F	
A,B	A,C	
A,C	A,E	
A,E	C,F	
	C,D	



# Broadphase benchmark

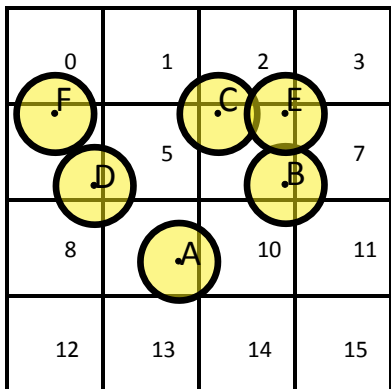
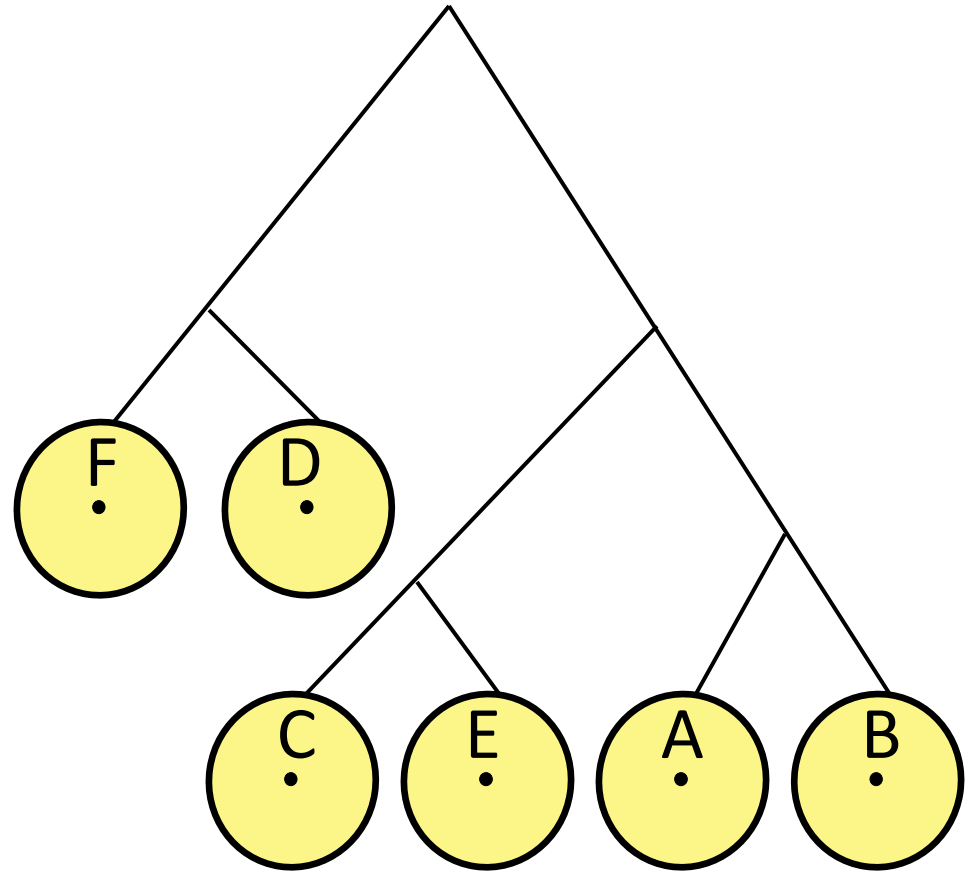
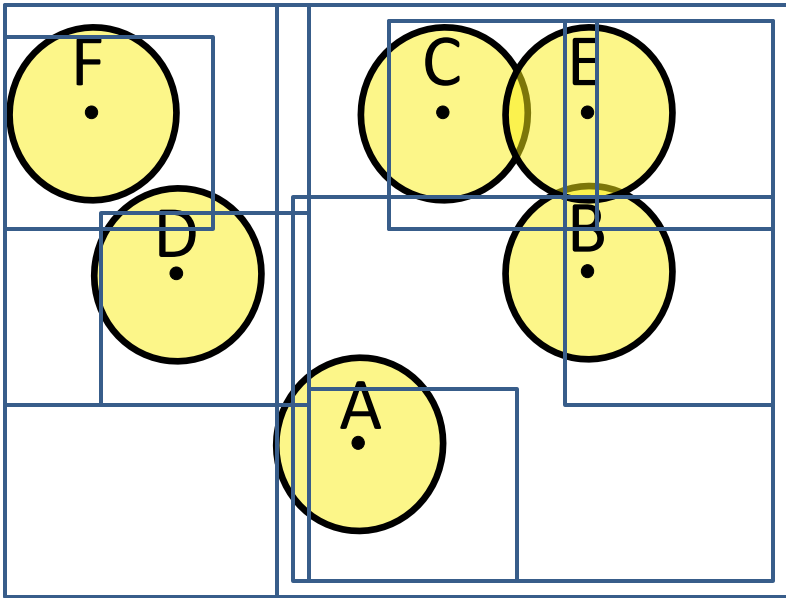
- Includes btCudaBroadphase
- Bullet SDK: Bullet/Extras/CDTestFramework



# From Particles to Rigid Bodies

	Particles	Rigid Bodies
World Transform	Position	Position and Orientation
Neighbor Search	Uniform Grid	Dynamic BVH tree
Compute Contacts	Sphere-Sphere	Generic Convex Closest Points, GJK
Static Geometry	Planes	Concave Triangle Mesh
Solving method	Jacobi	Projected Gauss Seidel

# Dynamic BVH Trees



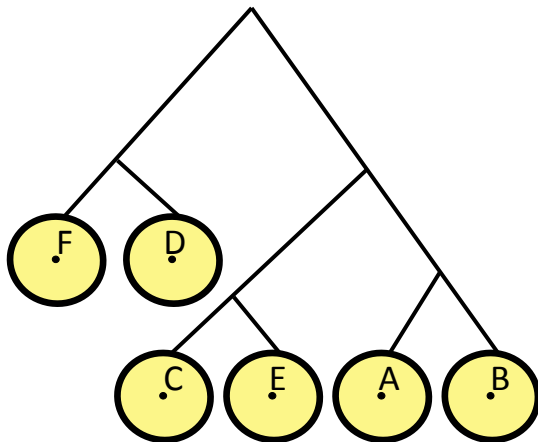
# Dynamic BVH tree acceleration structure

- Broadphase n-body neighbor search
- Ray and convex sweep test
- Concave triangle meshes
- Compound collision shapes

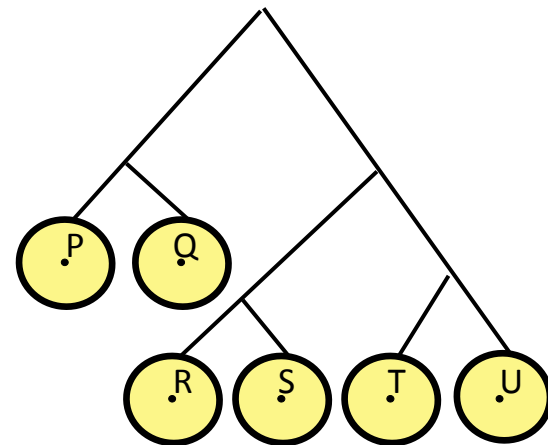
# Dynamic BVH tree Broadphase

- Keep two dynamic trees, one for moving objects, other for objects (sleeping/static)
- Find neighbor pairs:
  - Overlap M versus M and Overlap M versus S

S: Non-moving DBVT



M: Moving DBVT

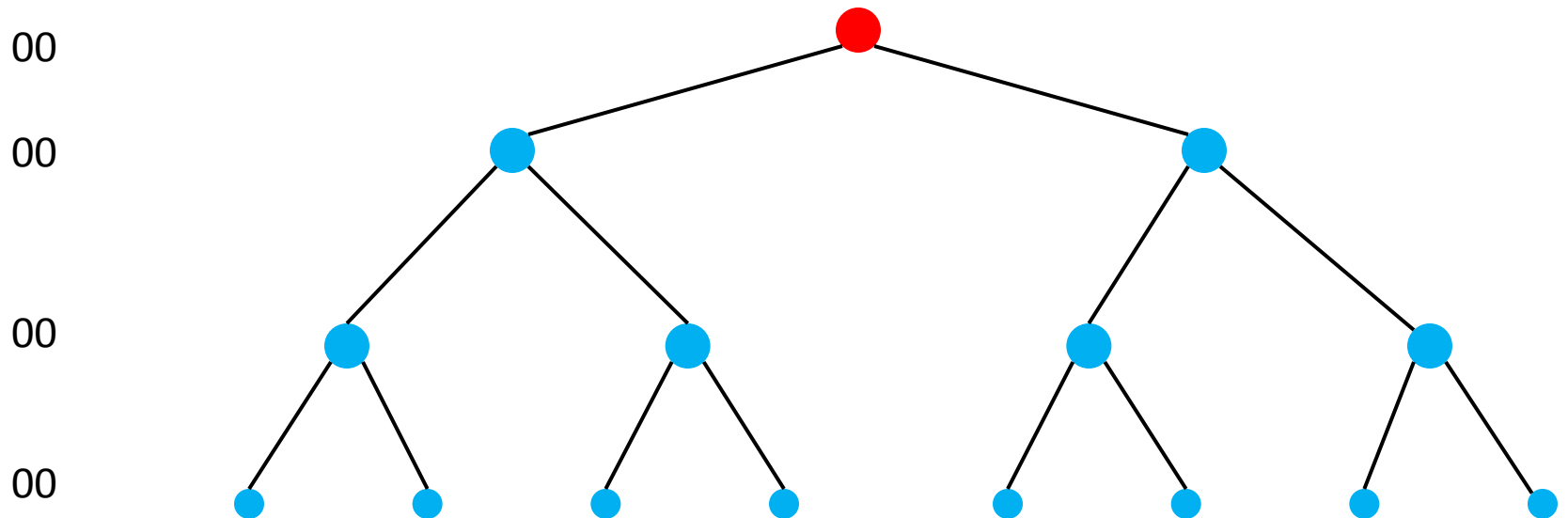


# DBVT Broadphase Optimizations

- Objects can move from one tree to the other
- Incrementally update, re-balance tree
- Tree update hard to parallelize
- Tree traversal can be parallelized on GPU
  - Idea proposed by Takahiro Harada at GDC 2009

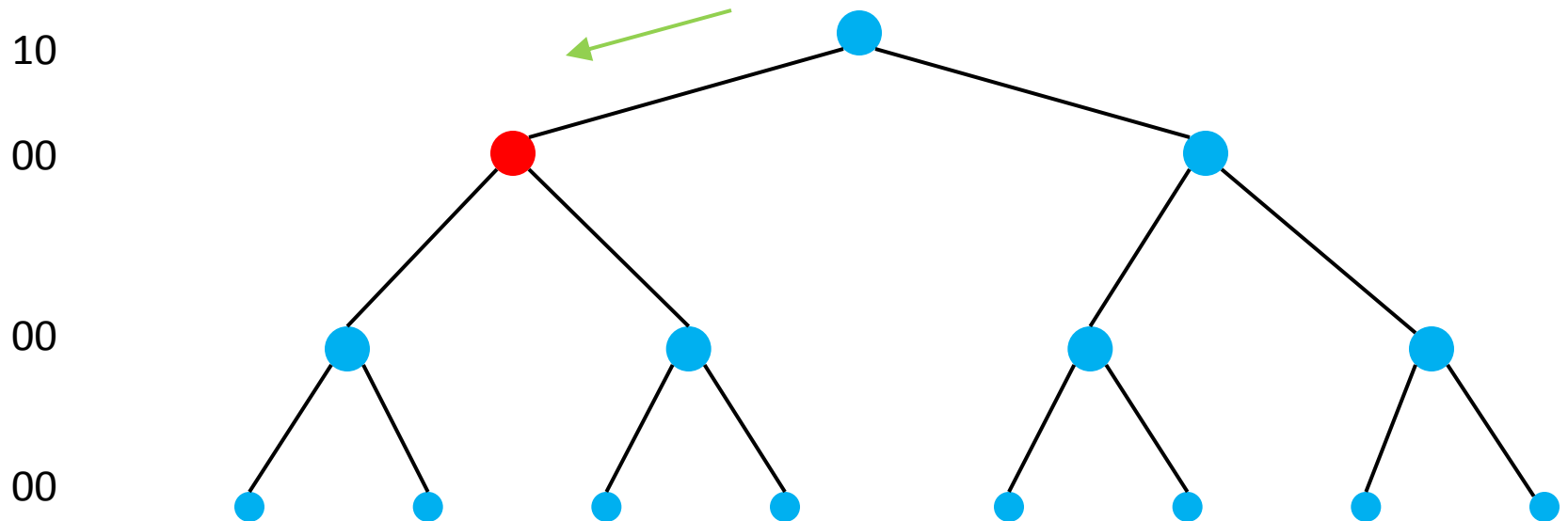
# Parallel GPU Tree Traversal using History Flags

- Alternative to recursive or stackless traversal



# Parallel GPU Tree Traversal using History Flags

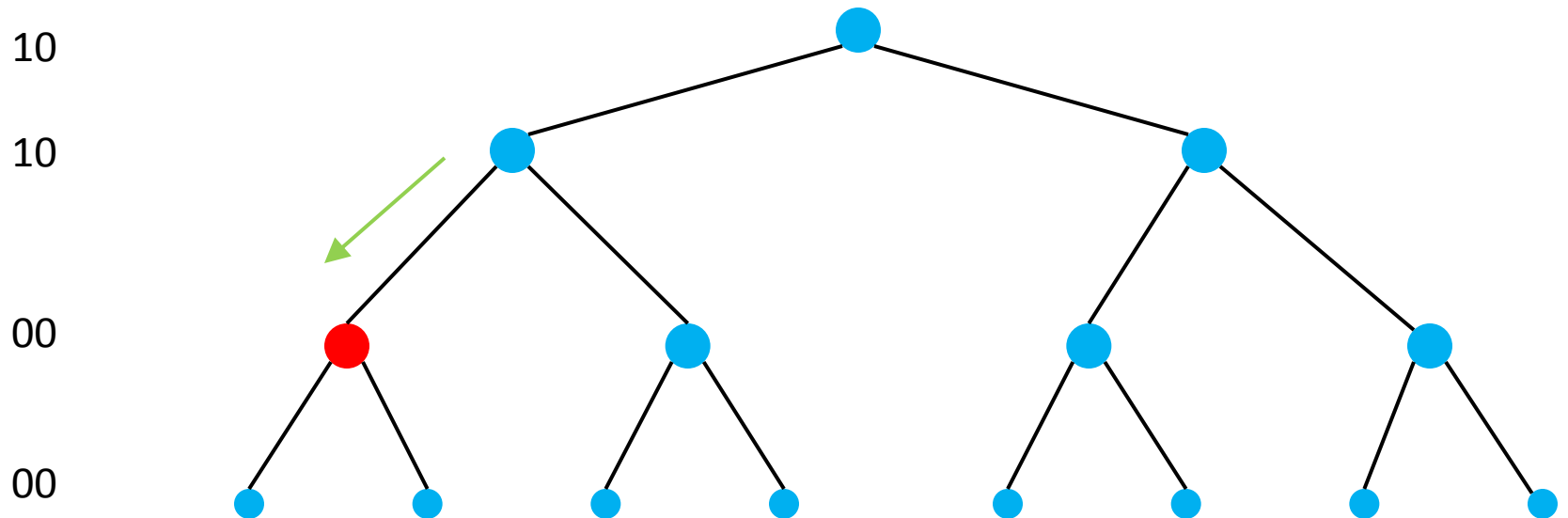
- 2 bits at each level indicating visited children





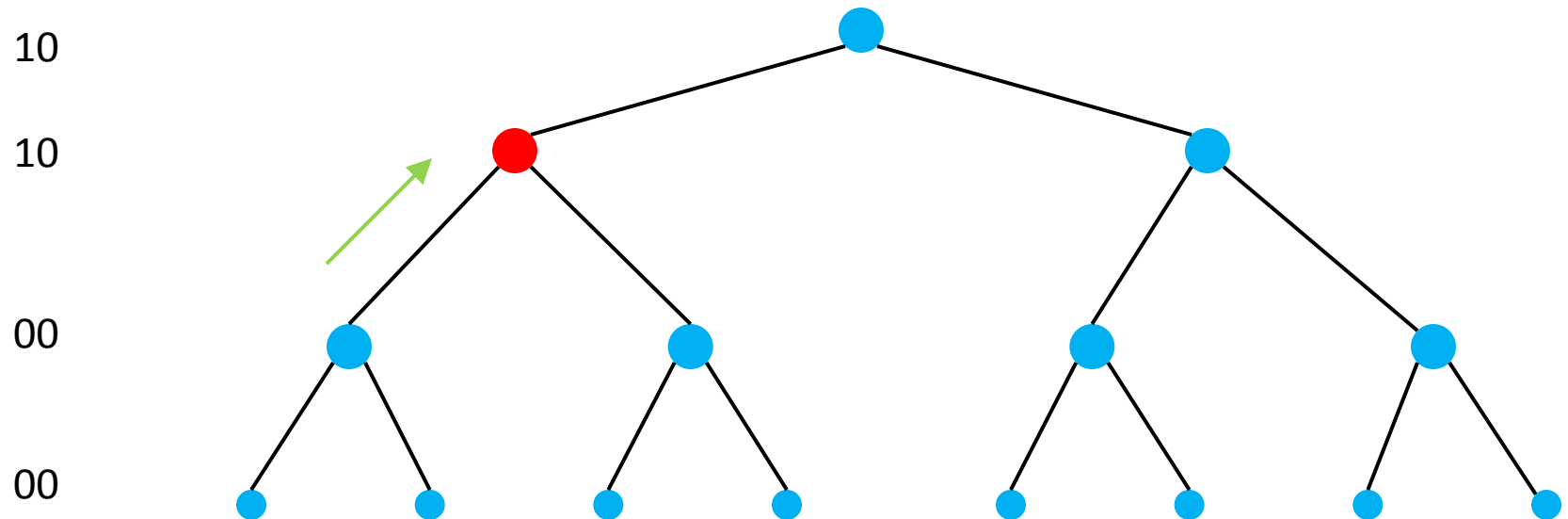
# Parallel GPU Tree Traversal using History Flags

- Set bit when descending into a child branch



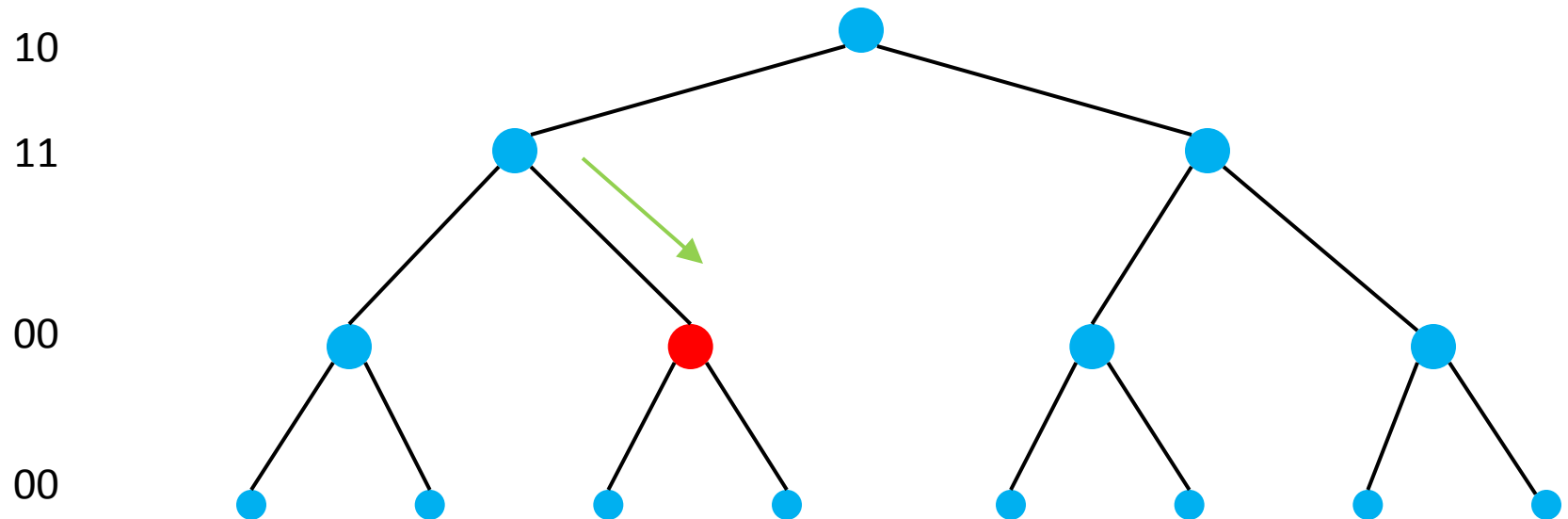
# Parallel GPU Tree Traversal using History Flags

- Reset bits when ascending up the tree



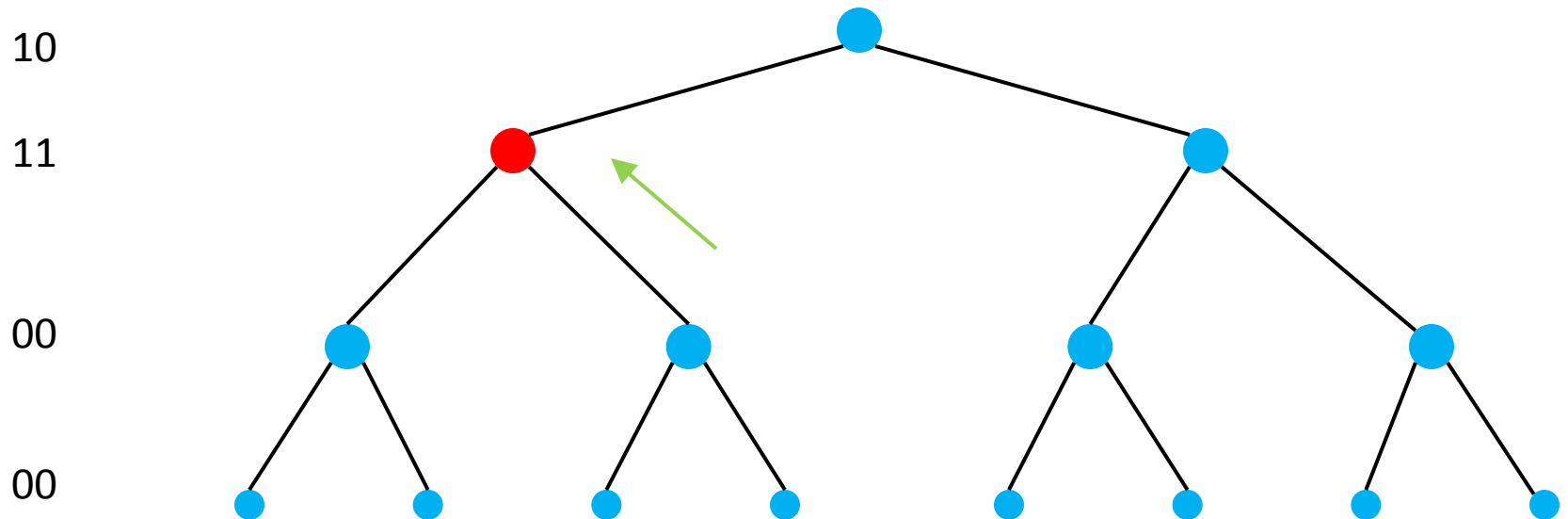
# Parallel GPU Tree Traversal using History Flags

- Requires only twice the tree depth bits



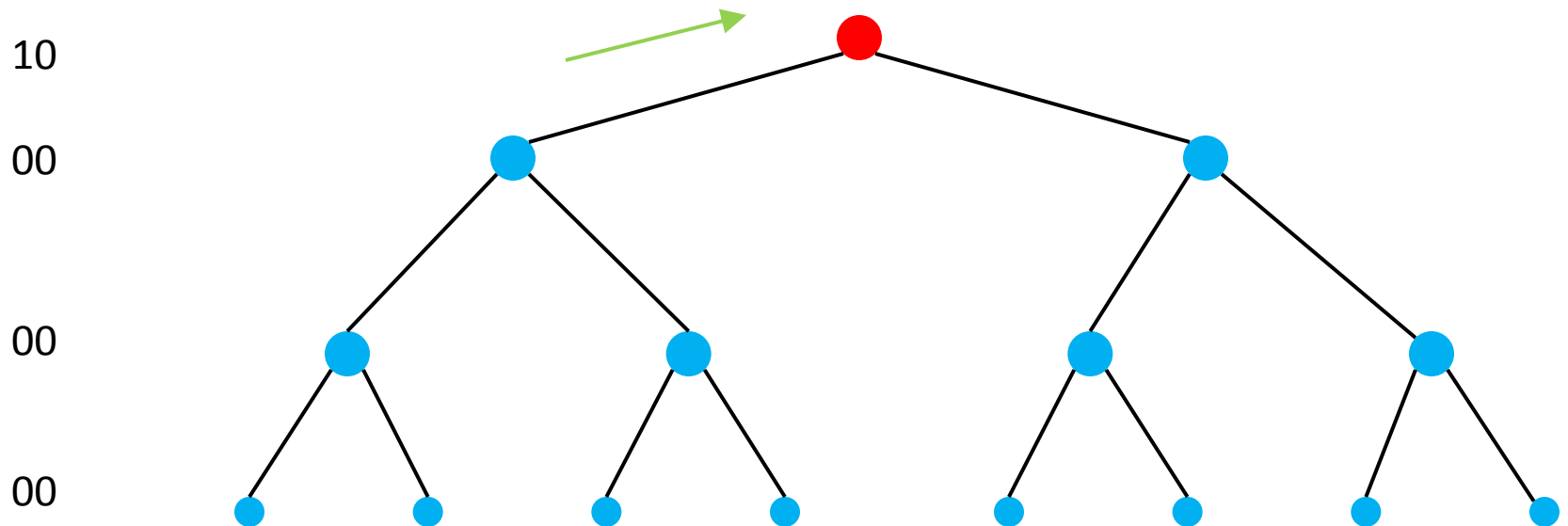
# Parallel GPU Tree Traversal using History Flags

- When both bits are set, ascend to parent



# Parallel GPU Tree Traversal using History Flags

- When both bits are set, ascend to parent

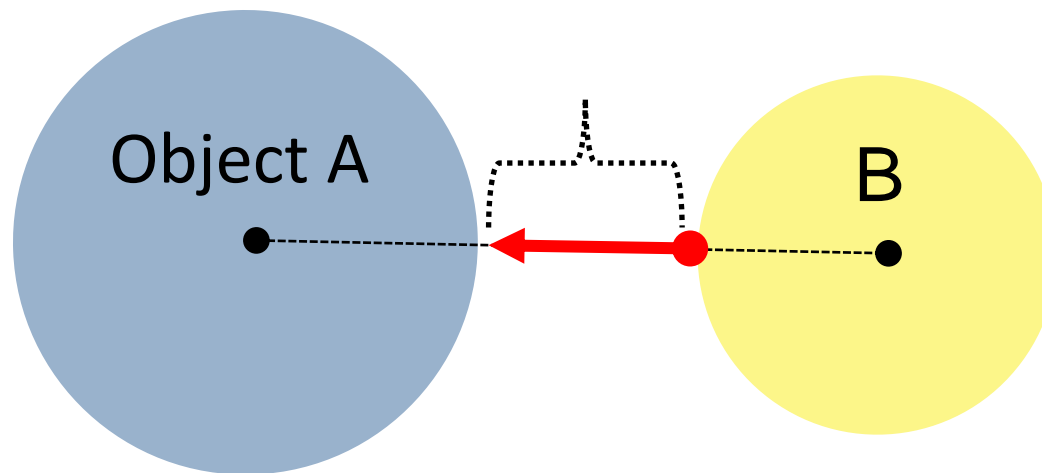


# History tree traversal

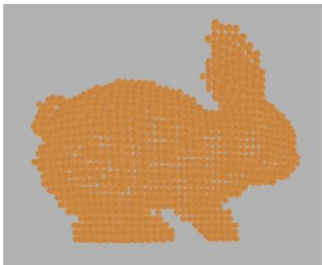
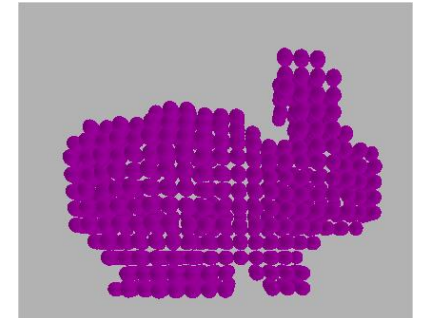
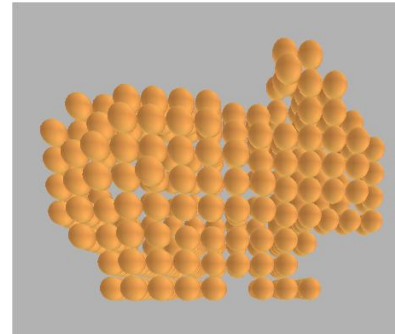
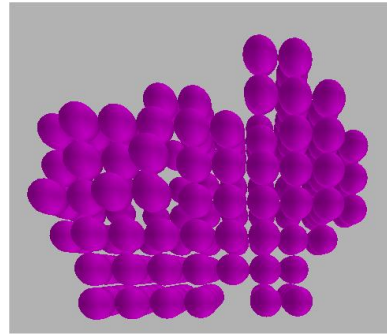
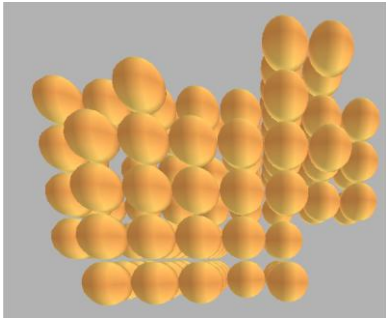
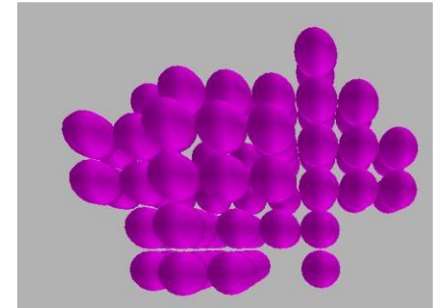
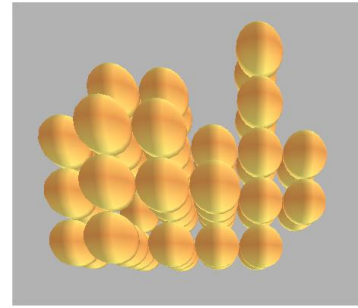
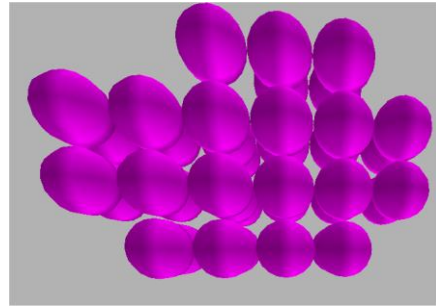
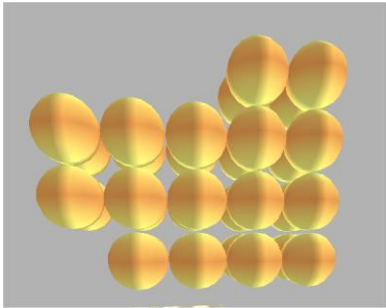
```
do{  
    if(Intersect(n->volume,volume)){  
        if(n->isinternal()) {  
            if (!historyFlags[curDepth].m_visitedLeftChild){  
                historyFlags[curDepth].m_visitedLeftChild = 1;  
                n = n->childs[0];  
                curDepth++;  
                continue;}  
            if (!historyFlags[curDepth].m_visitedRightChild){  
                historyFlags[curDepth].m_visitedRightChild = 1;  
                n = n->childs[1];  
                curDepth++;  
                continue;}  
        }  
        else  
            policy.Process(n);  
    }  
    n = n->parent;  
    historyFlags[curDepth].m_visitedLeftChild = 0;  
    historyFlags[curDepth].m_visitedRightChild = 0;  
    curDepth--;  
} while (curDepth);
```

# Find contact points

- Closest points, normal and distance
- Convention: positive distance  $\rightarrow$  separation
- Contact normal points from B to A

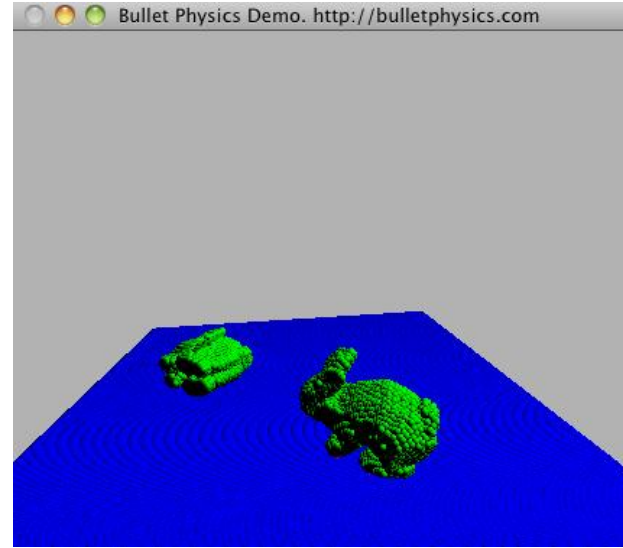
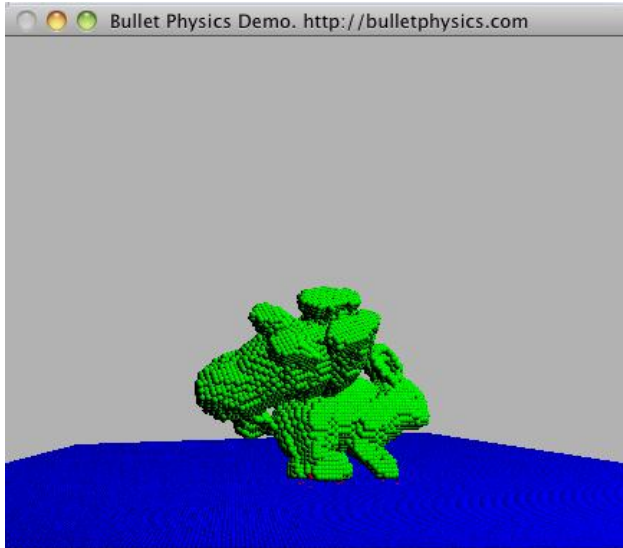
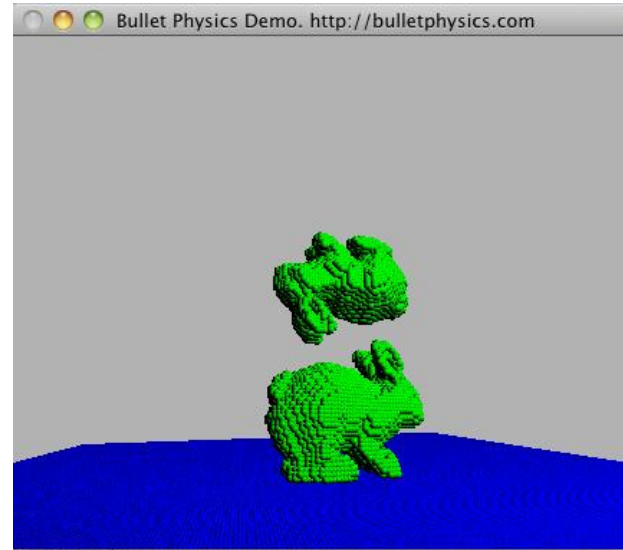
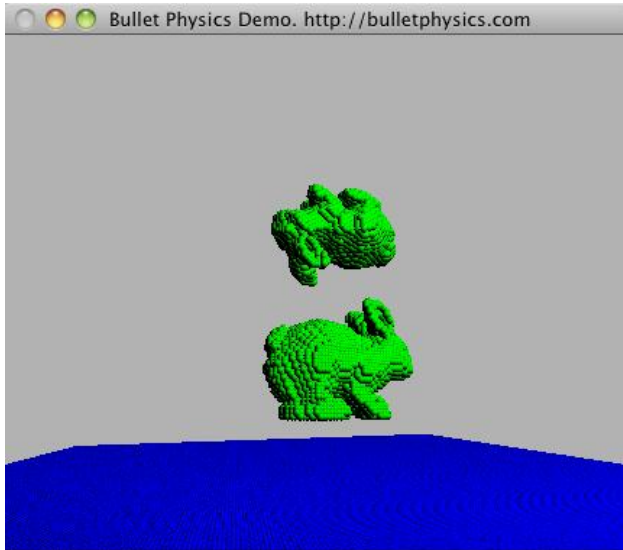


# Voxelizing objects



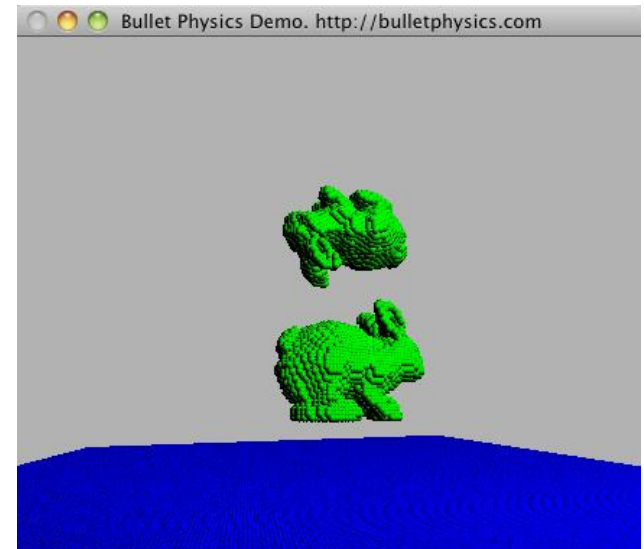


# OpenCL Rigid Particle Bunnies

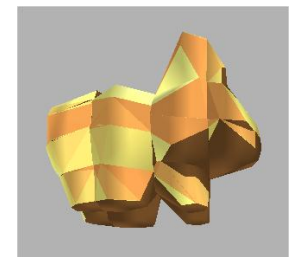
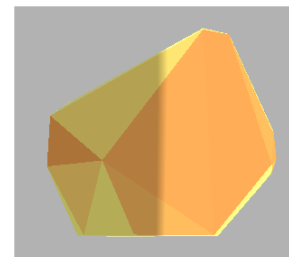
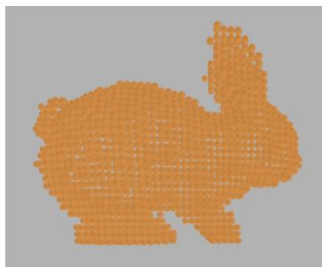
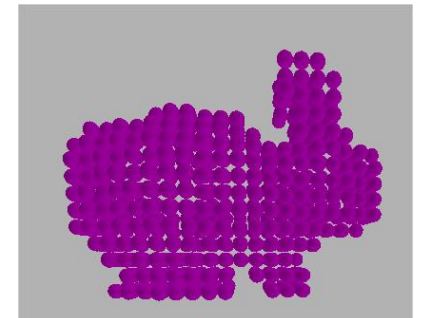
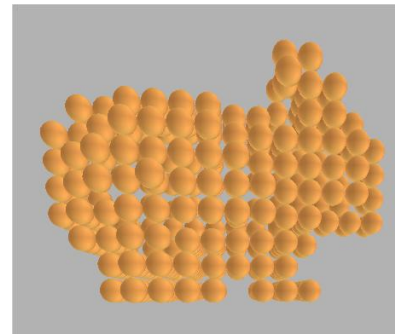
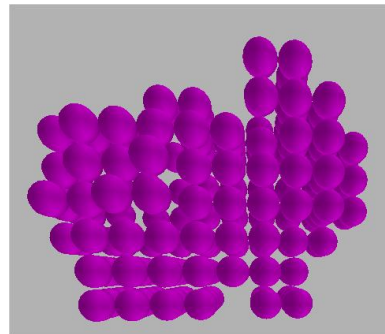
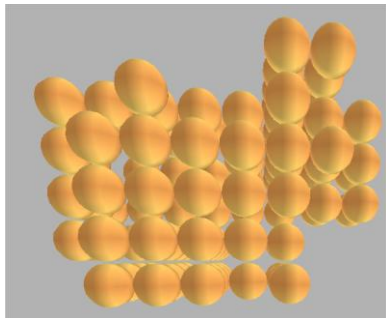
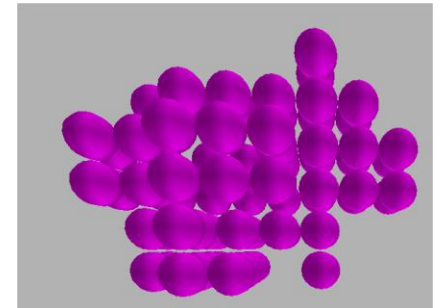
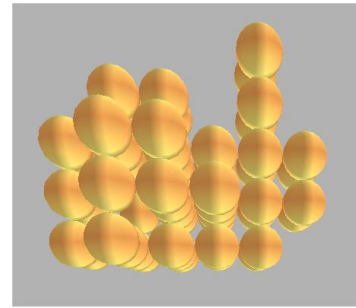
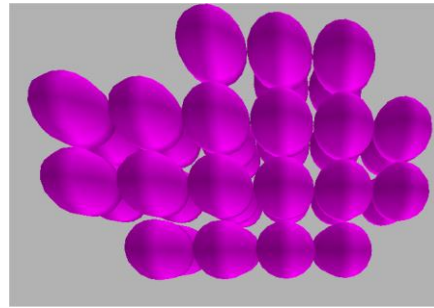
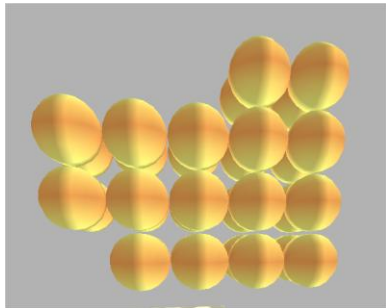


# Broadphase

- The bunny demo broadphase has entries for each particle to avoid  $n^2$  tests
- Many sphere-sphere contact pairs between two rigid bunnies
- Uniform Grid is not sufficient

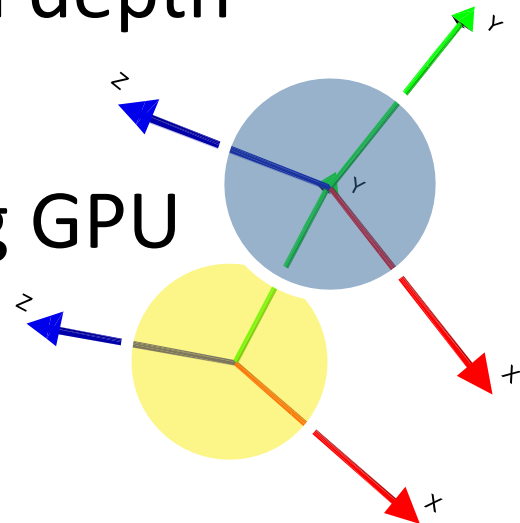


# Voxelizing objects



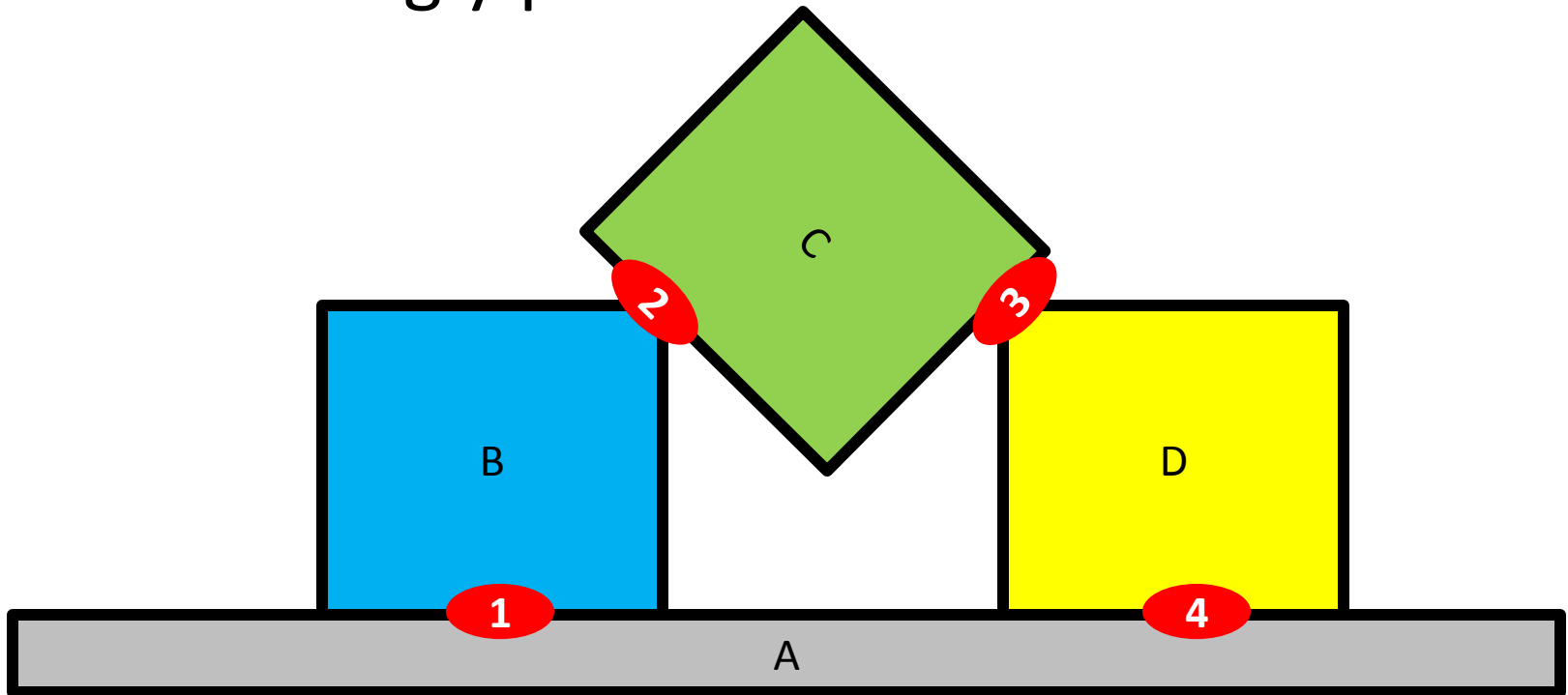
# General convex collision detection on GPU

- Bullet uses hybrid GJK algorithm with EPA
- GJK convex collision detection fits current GPU
- EPA penetration depth harder to port to GPU
  - Larger code size, dynamic data structures
- Instead of EPA, sample penetration depth
  - Using support mapping
- Support map can be sampled using GPU hardware



# Parallelizing Constraint Solver

- Projected Gauss Seidel iterations are not embarrassingly parallel

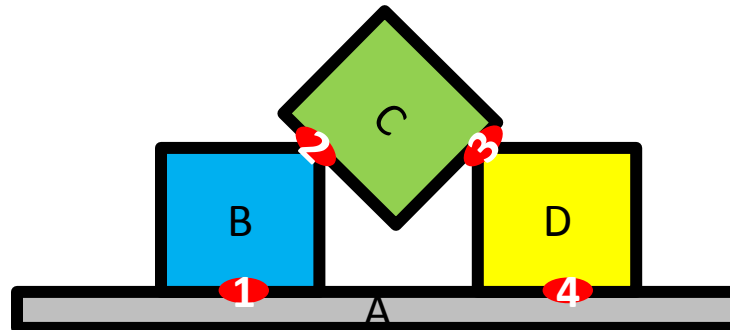


# Reordering constraint batches

A	B	C	D
1	1		
	2	2	
		3	3
4			4

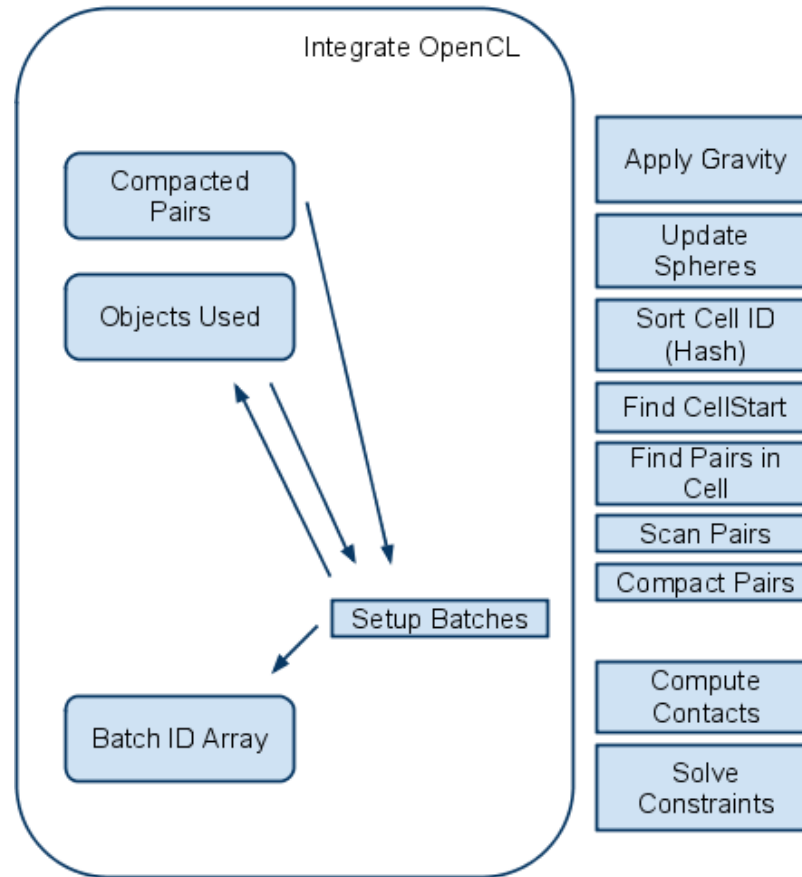


A	B	C	D
1	1	3	3
4	2	2	4



# Creating Parallel Batches

DEVICE



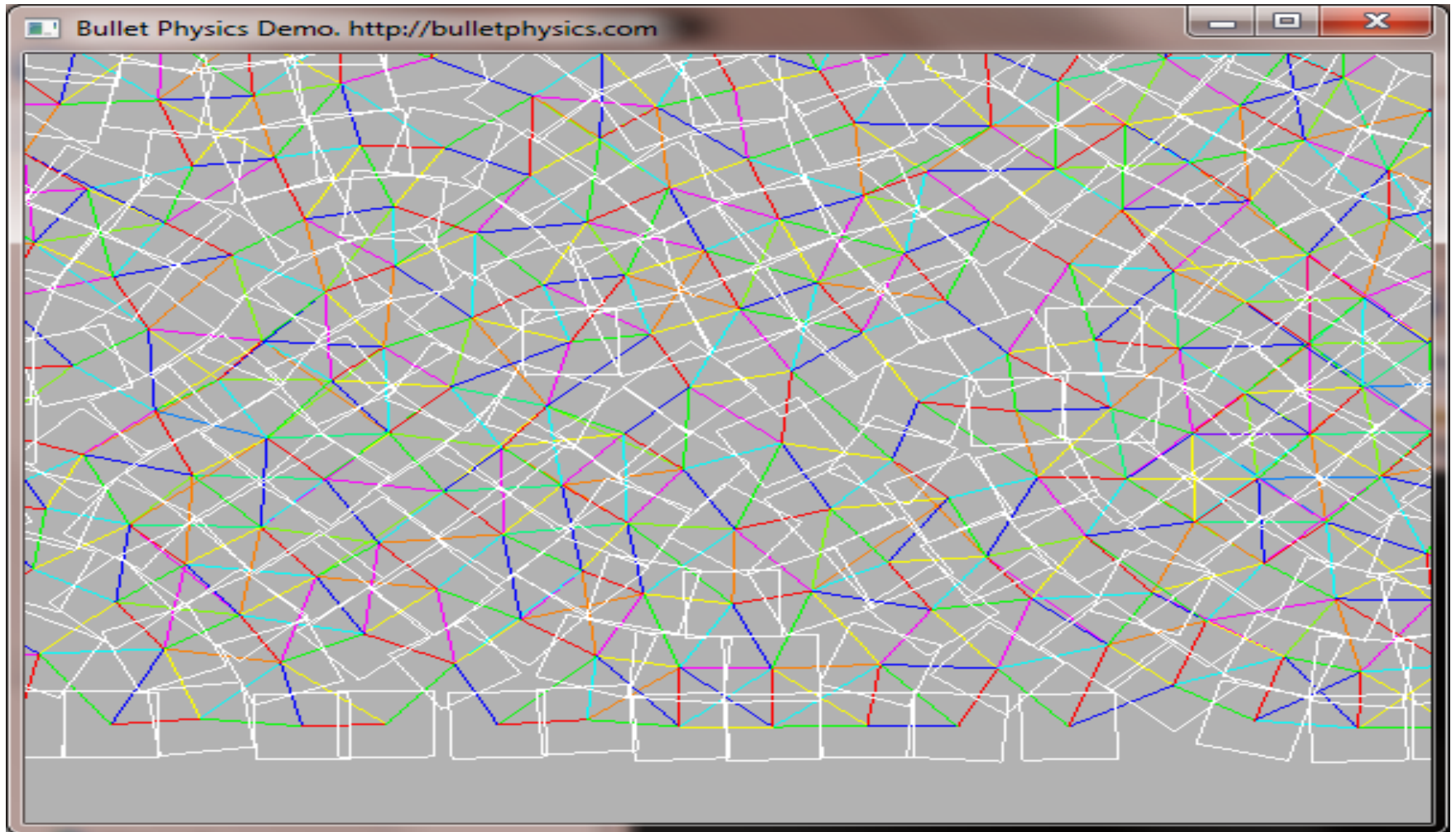
# OpenCL kernel Setup Batches

```
__kernel void kSetupBatches(...)
{
    int index = get_global_id(0);
    int currPair = index;
    int objIdA = pPairIds[currPair * 2].x;
    int objIdB = pPairIds[currPair * 2].y;
    int batchId = pPairIds[currPair * 2 + 1].x;
    int localWorkSz = get_local_size(0);
    int localIdx = get_local_id(0);
    for(int i = 0; i < localWorkSz; i++)
    {
        if((i==localIdx) &&(batchId < 0) &&(pObjUsed[objIdA]<0) &&(pObjUsed[objIdB]<0))
        {
            if(pObjUsed[objIdA] == -1)
                pObjUsed[objIdA] = index;
            if(pObjUsed[objIdB] == -1)
                pObjUsed[objIdB] = index;
        }

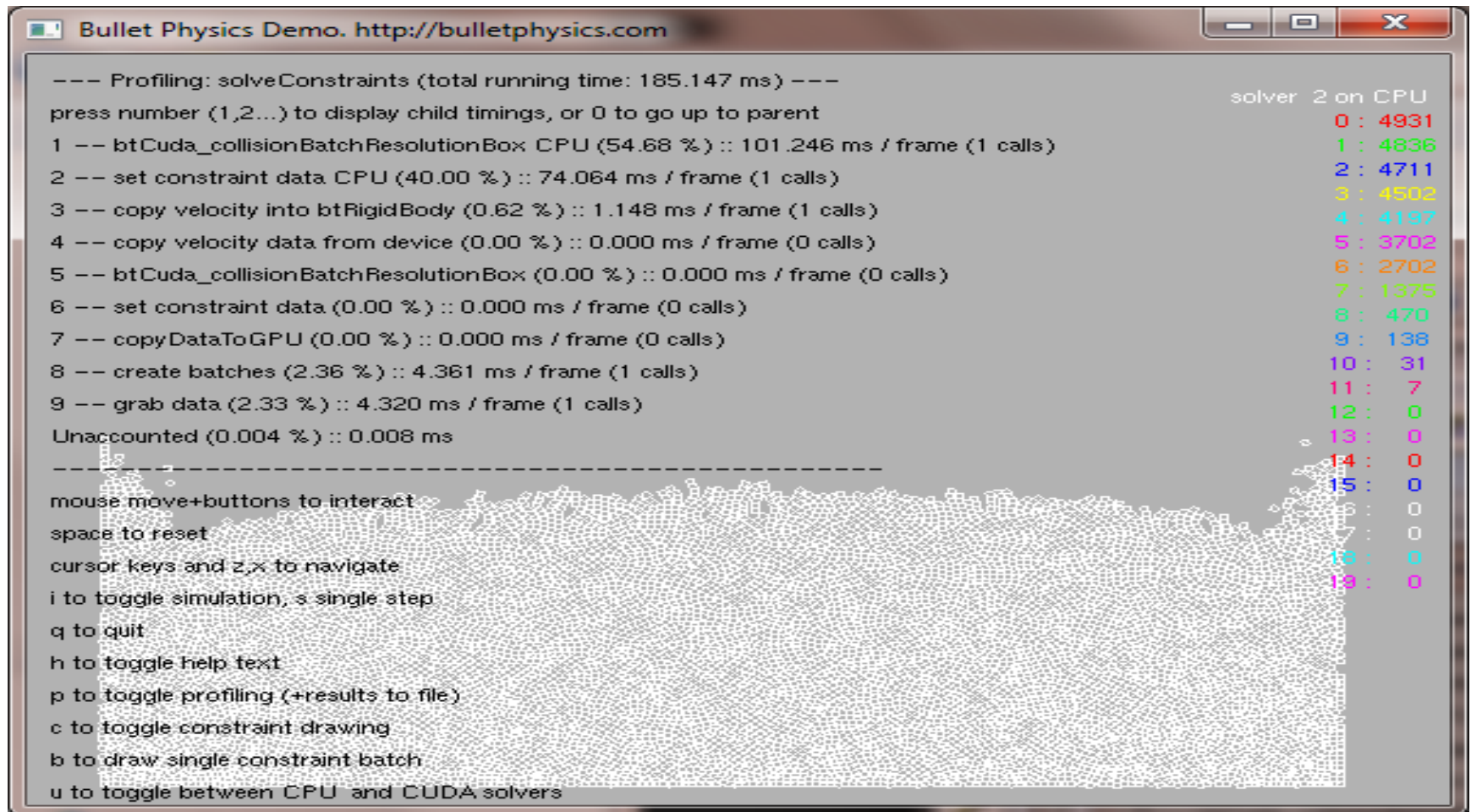
        barrier(CLK_GLOBAL_MEM_FENCE);
    }
}
```



# Colored Batches



# CPU 3Ghz single thread, 2D, 185ms



The screenshot shows a window titled "Bullet Physics Demo. http://bulletphysics.com". The main content is a text-based profiling report and a 2D simulation scene. The profiling report shows the total running time for solving constraints is 185.147 ms. The scene is a 2D simulation with many small objects and constraints.

```
--- Profiling: solveConstraints (total running time: 185.147 ms) ---
press number (1,2,...) to display child timings, or 0 to go up to parent
1 -- btCuda_collisionBatchResolutionBox CPU (54.68 %) :: 101.246 ms / frame (1 calls)
2 -- set constraint data CPU (40.00 %) :: 74.064 ms / frame (1 calls)
3 -- copy velocity into btRigidBody (0.62 %) :: 1.148 ms / frame (1 calls)
4 -- copy velocity data from device (0.00 %) :: 0.000 ms / frame (0 calls)
5 -- btCuda_collisionBatchResolutionBox (0.00 %) :: 0.000 ms / frame (0 calls)
6 -- set constraint data (0.00 %) :: 0.000 ms / frame (0 calls)
7 -- copyDataToGPU (0.00 %) :: 0.000 ms / frame (0 calls)
8 -- create batches (2.36 %) :: 4.361 ms / frame (1 calls)
9 -- grab data (2.33 %) :: 4.320 ms / frame (1 calls)
Unaccounted (0.004 %) :: 0.008 ms
```

-----

mouse move+buttons to interact  
space to reset  
cursor keys and z,x to navigate  
i to toggle simulation, s single step  
q to quit  
h to toggle help text  
p to toggle profiling (+results to file)  
c to toggle constraint drawing  
b to draw single constraint batch  
u to toggle between CPU and CUDA solvers

solver 2 on CPU	
0	: 4931
1	: 4836
2	: 4711
3	: 4502
4	: 4197
5	: 3702
6	: 2702
7	: 1375
8	: 470
9	: 138
10	: 31
11	: 7
12	: 0
13	: 0
14	: 0
15	: 0
16	: 0
17	: 0
18	: 0
19	: 0

# Geforce 260 CUDA, 2D, 21ms

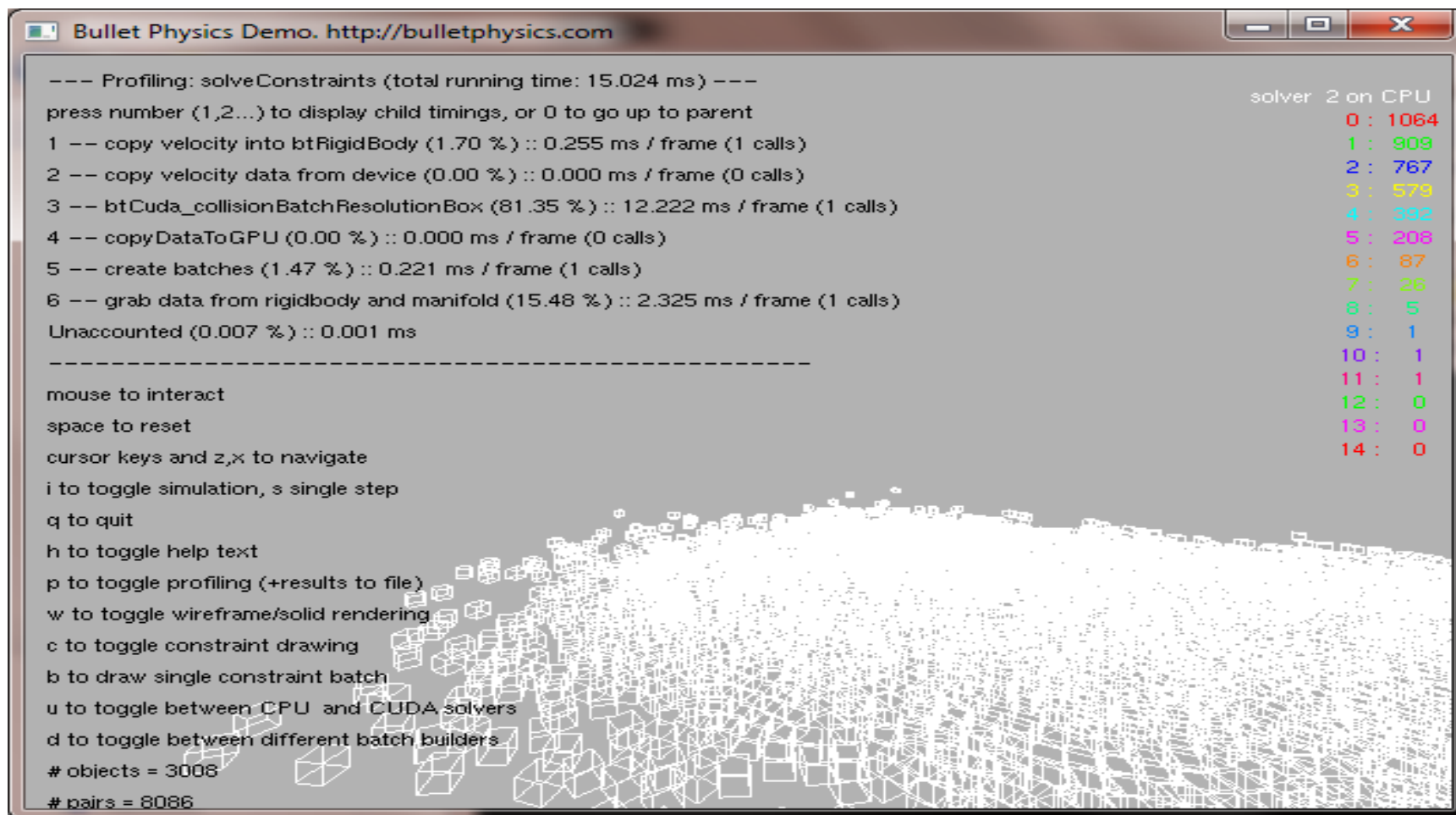
```
Bullet Physics Demo. http://bulletphysics.com

--- Profiling: solveConstraints (total running time: 21.978 ms) ---
press number (1,2,...) to display child timings, or 0 to go up to parent
1 -- btCuda_collisionBatchResolutionBox CPU (0.00 %) :: 0.000 ms / frame (0 calls)
2 -- set constraint data CPU (0.00 %) :: 0.000 ms / frame (0 calls)
3 -- copy velocity into btRigidBody (9.91 %) :: 2.178 ms / frame (1 calls)
4 -- copy velocity data from device (2.63 %) :: 0.579 ms / frame (1 calls)
5 -- btCuda_collisionBatchResolutionBox (41.56 %) :: 9.133 ms / frame (1 calls)
6 -- set constraint data (11.24 %) :: 2.470 ms / frame (1 calls)
7 -- copyDataToGPU (5.27 %) :: 1.158 ms / frame (1 calls)
8 -- create batches (17.34 %) :: 3.812 ms / frame (1 calls)
9 -- grab data (12.01 %) :: 2.639 ms / frame (1 calls)
Unaccounted (0.041 %) :: 0.009 ms

-----
mouse move+buttons to interact
space to reset
cursor keys and z,x to navigate
i to toggle simulation, s single step
q to quit
h to toggle help text
p to toggle profiling (+results to file)
c to toggle constraint drawing
b to draw single constraint batch
u to toggle between CPU and CUDA solvers
```

solver 2 on CUDA	
0	: 4930
1	: 4829
2	: 4710
3	: 4511
4	: 4229
5	: 3683
6	: 2727
7	: 1432
8	: 505
9	: 142
10	: 39
11	: 8
12	: 0
13	: 0
14	: 0
15	: 0
16	: 0
17	: 0
18	: 0
19	: 0

# CPU 3Ghz single thread, 3D, 12ms



Bullet Physics Demo. <http://bulletphysics.com>

--- Profiling: solveConstraints (total running time: 15.024 ms) ---

press number (1,2,...) to display child timings, or 0 to go up to parent

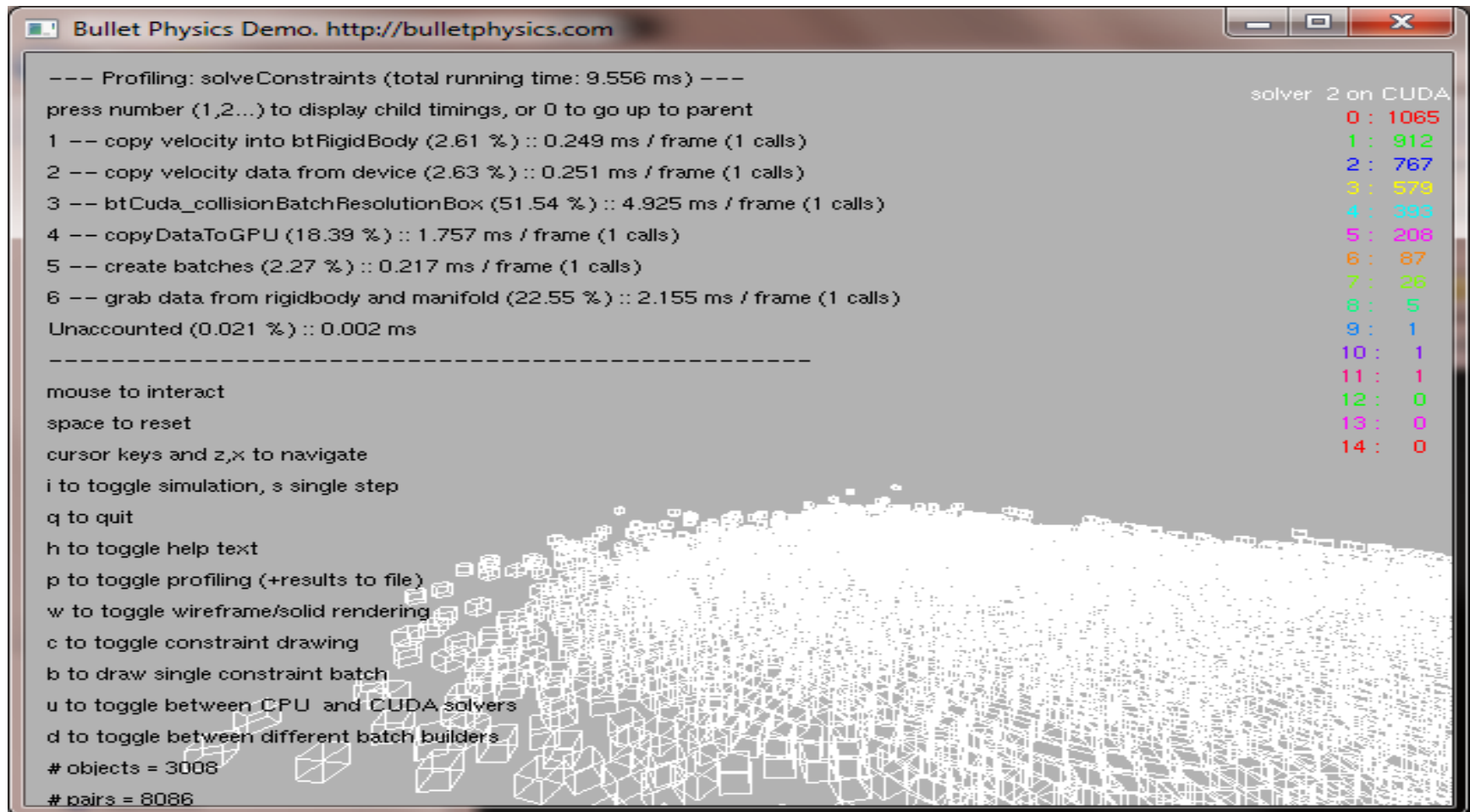
1 -- copy velocity into btRigidBody (1.70 %) :: 0.255 ms / frame (1 calls)	0 : 1064
2 -- copy velocity data from device (0.00 %) :: 0.000 ms / frame (0 calls)	1 : 909
3 -- btCuda_collisionBatchResolutionBox (81.35 %) :: 12.222 ms / frame (1 calls)	2 : 767
4 -- copyDataToGPU (0.00 %) :: 0.000 ms / frame (0 calls)	3 : 579
5 -- create batches (1.47 %) :: 0.221 ms / frame (1 calls)	4 : 392
6 -- grab data from rigidbody and manifold (15.48 %) :: 2.325 ms / frame (1 calls)	5 : 208
Unaccounted (0.007 %) :: 0.001 ms	6 : 87
	7 : 26
	8 : 5
	9 : 1
	10 : 1
	11 : 1
	12 : 0
	13 : 0
	14 : 0

-----

mouse to interact  
space to reset  
cursor keys and z,x to navigate  
i to toggle simulation, s single step  
q to quit  
h to toggle help text  
p to toggle profiling (+results to file)  
w to toggle wireframe/solid rendering  
c to toggle constraint drawing  
b to draw single constraint batch  
u to toggle between CPU and CUDA solvers  
d to toggle between different batch builders

# objects = 3008  
# pairs = 8086

# Geforce 260 CUDA, 3D, 4.9ms



Bullet Physics Demo. <http://bulletphysics.com>

--- Profiling: solveConstraints (total running time: 9.556 ms) ---

press number (1,2,...) to display child timings, or 0 to go up to parent

1 -- copy velocity into btRigidBody (2.61 %) :: 0.249 ms / frame (1 calls)	0 : 1065
2 -- copy velocity data from device (2.63 %) :: 0.251 ms / frame (1 calls)	1 : 912
3 -- btCuda_collisionBatchResolutionBox (51.54 %) :: 4.925 ms / frame (1 calls)	2 : 767
4 -- copyDataToGPU (18.39 %) :: 1.757 ms / frame (1 calls)	3 : 579
5 -- create batches (2.27 %) :: 0.217 ms / frame (1 calls)	4 : 393
6 -- grab data from rigidbody and manifold (22.55 %) :: 2.155 ms / frame (1 calls)	5 : 208
Unaccounted (0.021 %) :: 0.002 ms	6 : 87
	7 : 26
	8 : 5
	9 : 1
	10 : 1
	11 : 1
	12 : 0
	13 : 0
	14 : 0

-----

mouse to interact  
space to reset  
cursor keys and z,x to navigate  
i to toggle simulation, s single step  
q to quit  
h to toggle help text  
p to toggle profiling (+results to file)  
w to toggle wireframe/solid rendering  
c to toggle constraint drawing  
b to draw single constraint batch  
u to toggle between CPU and CUDA solvers  
d to toggle between different batch builders

# objects = 3008  
# pairs = 8086

# OpenCL Implementation

- Available in SVN branches/OpenCL
  - <http://bullet.googlecode.com>
- Tested various OpenCL implementations
  - NVidia GPU on Windows PC
  - Apple Snow Leopard on Geforce GPU and CPU
  - Intel, AMD CPU, ATI GPU (available soon)
  - Generic CPU through MiniCL
    - OpenCL kernels compiled and linked as regular C
    - Multi-threaded or sequential for easier debugging

# Thanks!

- Questions?
- Visit the Physics Simulation Forum at
  - <http://bulletphysics.com>
- Email: `erwin_coumans@playstation.sony.com`