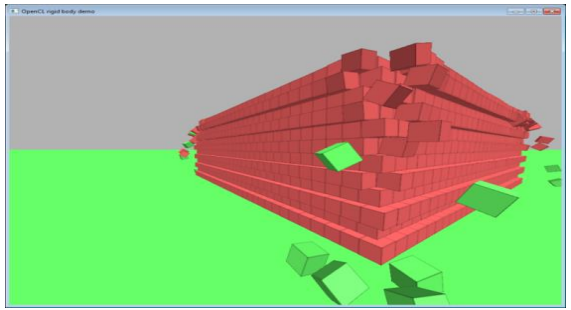
GPU Rigid Body Simulation

Erwin Coumans
Principal Engineer @ http://bulletphysics.org

GAME DEVELOPERS CONFERENCE
SAN FRANCISCO, CA
MARCH 25-29, 2013
EXPO DATES: MARCH 27-29
2013

Erwin Coumans

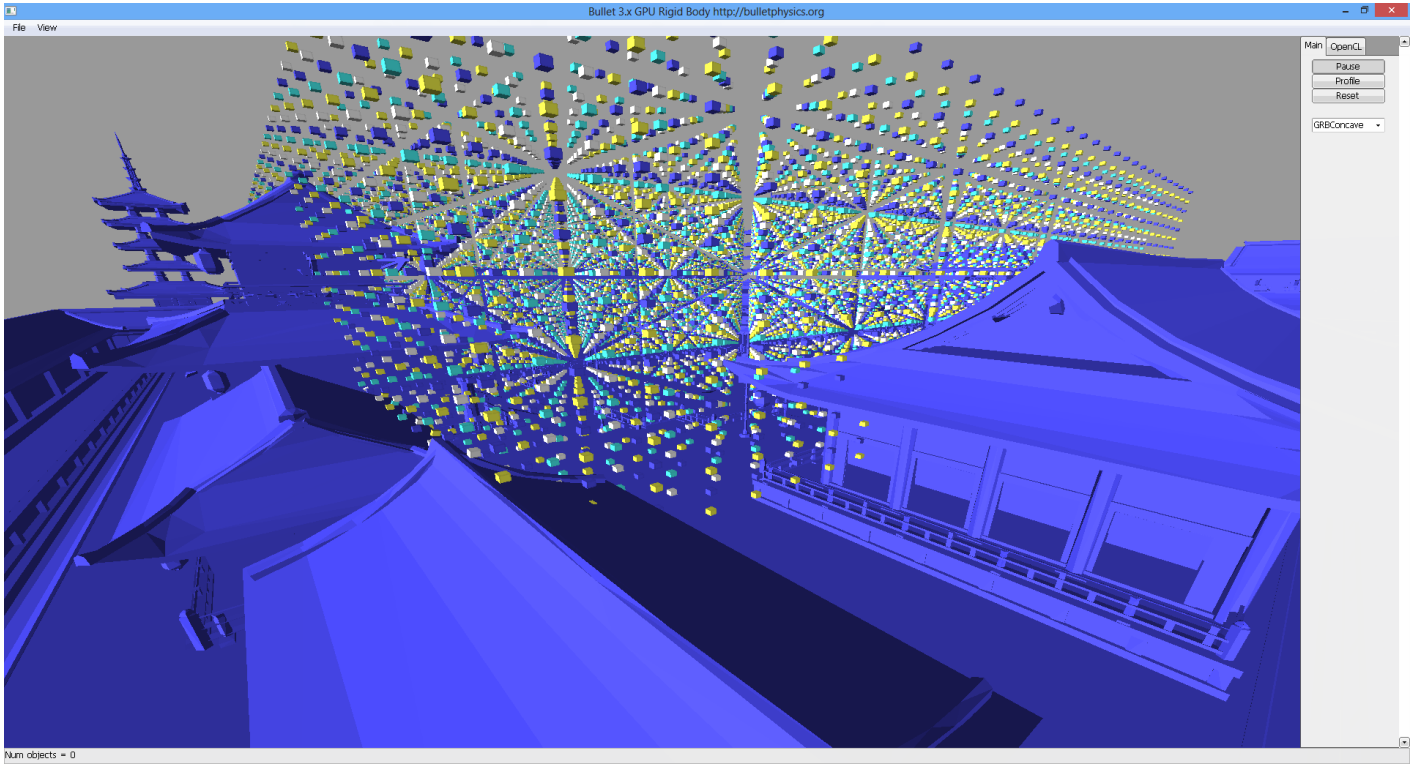
- Leading the Bullet Physics SDK project http://bulletphysics.org
- Doing GPGPU physics R&D at AMD, open source at http://github.com/erwincoumans/experiments
- Previously at Sony SCEA US R&D and Havok



GPU Cloth (2009)

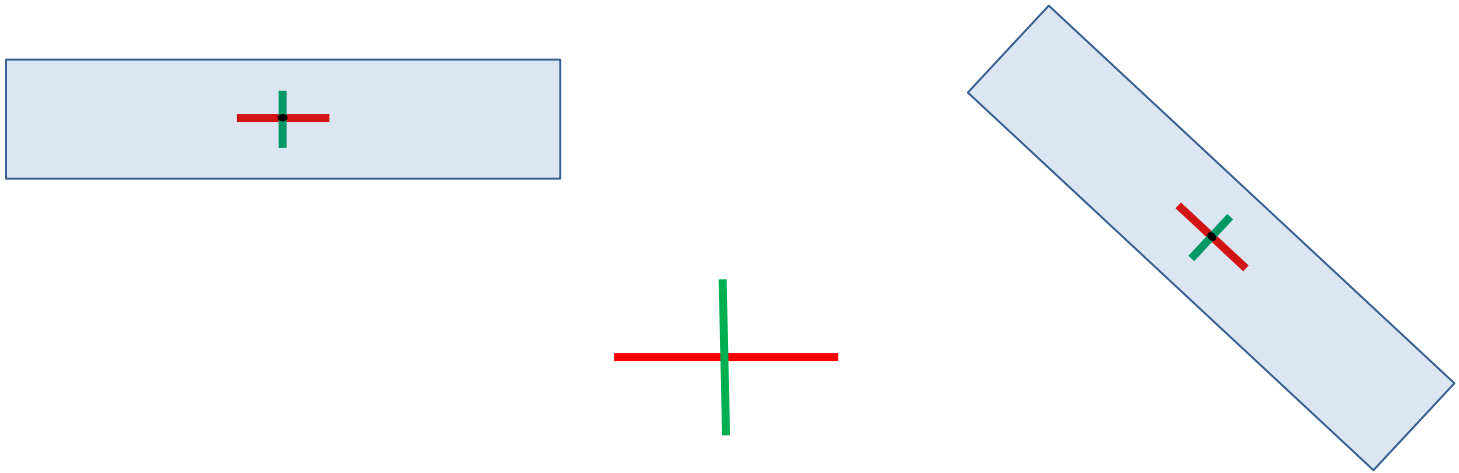
GPU Hair (2012/2013)

GPU Rigid Body (2008-2013)



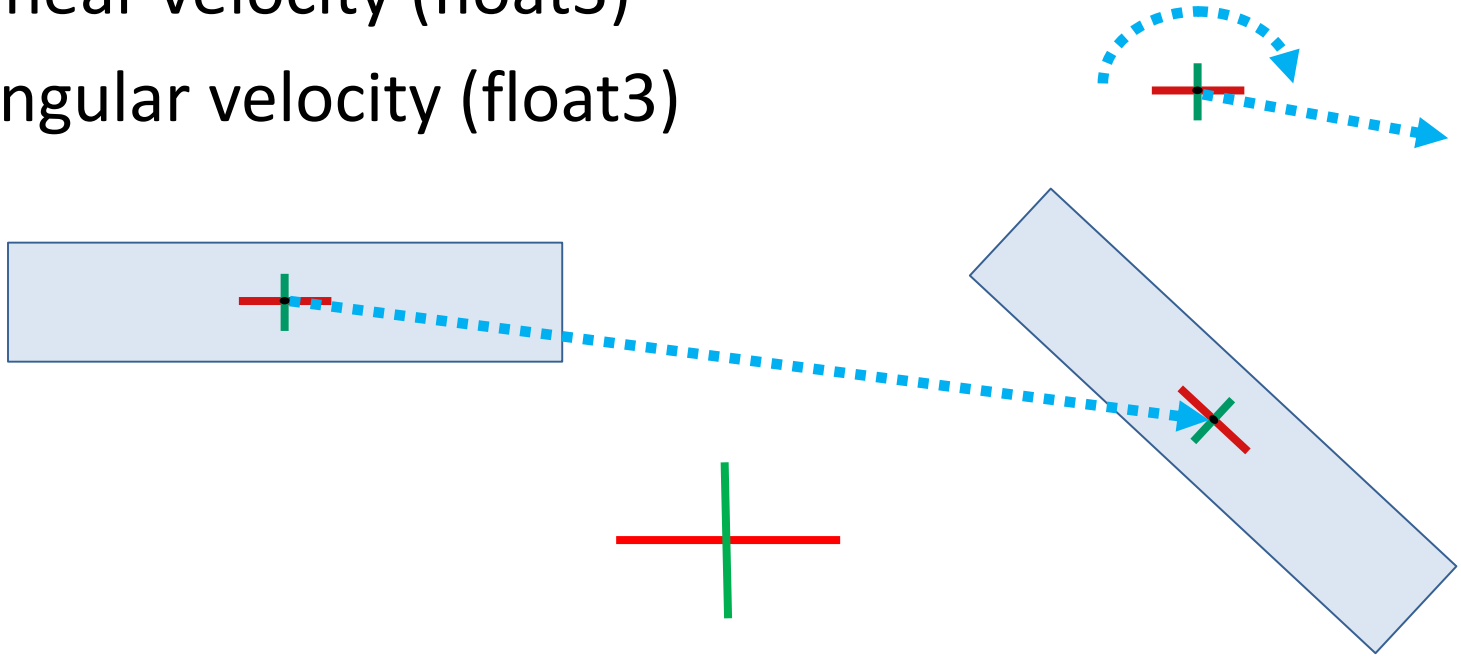
Rigid Bodies

- Position (Center of mass, float3)
- Orientation (Inertia basis frame, float4)



Updating the transform

- Linear velocity (float3)
- Angular velocity (float3)



Update Position in C/C++

```
void   integrateTransformsKernel(Body* bodies, int nodeID, float timeStep)
{
                        if( bodies[nodeID].m_invMass != 0.f)
                        {
                                                bodies[nodeID].m_pos +=  bodies[nodeID].m_linVel * timeStep;                        //linear velocity
                        }
}
```

Update Position in OpenCL™

```
__kernel void   integrateTransformsKernel( __global Body* bodies,const int numNodes, float timeStep)
{
                        int nodeID = get_global_id(0);
                        if( nodeID < numNodes && (bodies[nodeID].m_invMass != 0.f))
                        {
                                        bodies[nodeID].m_pos +=  bodies[nodeID].m_linVel * timeStep;                    //linear velocity
                        }
}
```

See opencl/gpu_rigidbody/kernels/integrateKernel.cl

Apply Gravity

```
__kernel void   integrateTransformsKernel( __global Body* bodies,const int numNodes, float timeStep, float angularDamping, float4 gravityAcceleration)
{
                        int nodeID = get_global_id(0);
                        if( nodeID < numNodes && (bodies[nodeID].m_invMass != 0.f))
                        {
                                                bodies[nodeID].m_pos +=  bodies[nodeID].m_linVel * timeStep;                    //linear velocity
                                                bodies[nodeID].m_linVel += gravityAcceleration * timeStep;                      //apply gravity
                        }
}
```

See opencl/gpu_rigidbody/kernels/integrateKernel.cl

Update Orientation

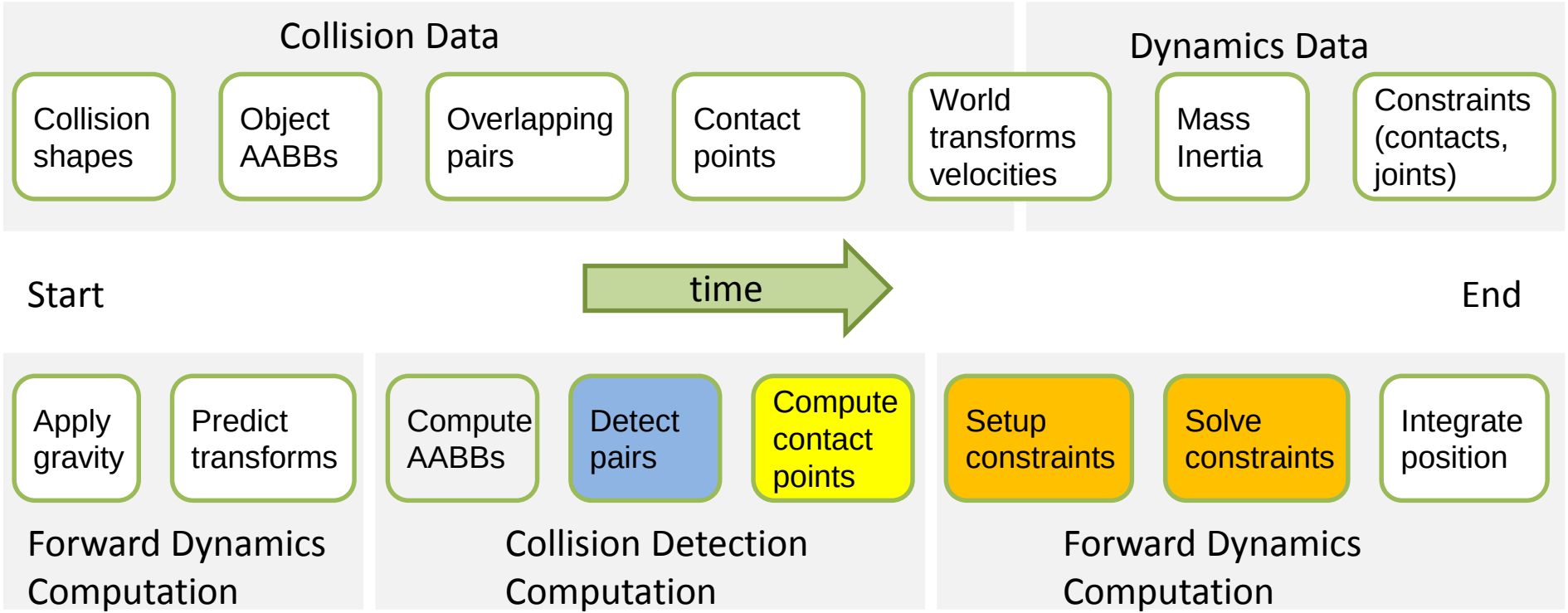
```
__kernel void   integrateTransformsKernel( __global Body* bodies,const int numNodes, float timeStep, float angularDamping, float4 gravityAcceleration)
{
        int nodeID = get_global_id(0);
        if( nodeID < numNodes && (bodies[nodeID].m_invMass != 0.f))
        {
                bodies[nodeID].m_pos +=  bodies[nodeID].m_linVel * timeStep;                    //linear velocity
                bodies[nodeID].m_linVel += gravityAcceleration * timeStep;                      //apply gravity
                 float4 angvel = bodies[nodeID].m_angVel;                                       //angular velocity
                bodies[nodeID].m_angVel *= angularDamping;                                      //add some angular damping
                 float4 axis;
                float fAngle = native_sqrt(dot(angvel, angvel));
                if(fAngle*timeStep> BT_GPU_ANGULAR_MOTION_THRESHOLD)                            //limit the angular motion
                        fAngle = BT_GPU_ANGULAR_MOTION_THRESHOLD / timeStep;
                if(fAngle < 0.001f)
                        axis = angvel * (0.5f*timeStep-(timeStep*timeStep*timeStep)*0.020833333333f * fAngle * fAngle);
                else
                        axis = angvel * ( native_sin(0.5f * fAngle * timeStep) / fAngle);
                float4 dorn = axis;
                dorn.w = native_cos(fAngle * timeStep * 0.5f);
                float4 orn0 = bodies[nodeID].m_quat;
                float4 predictedOrn = quatMult(dorn, orn0);
                predictedOrn = quatNorm(predictedOrn);
                bodies[nodeID].m_quat=predictedOrn;                                             //update the orientation
        }
}
```

See opencl/gpu_rigidbody/kernels/integrateKernel.cl

Update Transforms, Host Setup

```
ciErrNum = clSetKernelArg(g_integrateTransformsKernel, 0, sizeof(cl_mem), &bodies);
ciErrNum = clSetKernelArg(g_integrateTransformsKernel, 1, sizeof(int), &numBodies);
ciErrNum = clSetKernelArg(g_integrateTransformsKernel, 1, sizeof(float), &deltaTime);
ciErrNum = clSetKernelArg(g_integrateTransformsKernel, 1, sizeof(float), &angularDamping);
ciErrNum = clSetKernelArg(g_integrateTransformsKernel, 1, sizeof(float4), &gravityAcceleration);

size_t workGroupSize = 64;
size_t     numWorkItems = workGroupSize*((m_numPhysicsInstances + (workGroupSize)) / workGroupSize);
if (workGroupSize>numWorkItems)
        workGroupSize=numWorkItems;
ciErrNum = clEnqueueNDRangeKernel(g_cqCommandQue, g_integrateTransformsKernel, 1, NULL, &numWorkItems, &workGroupSize,0 ,0 ,0);
```

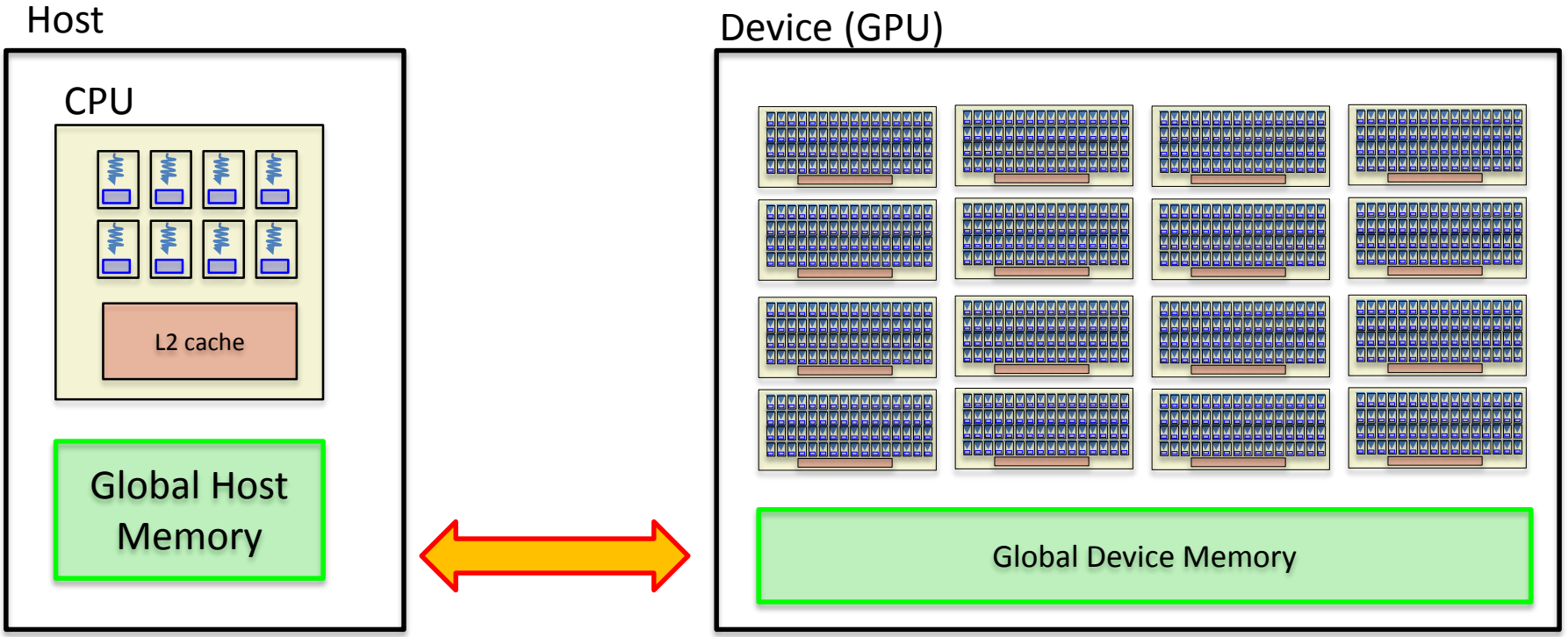
Physics pipeline



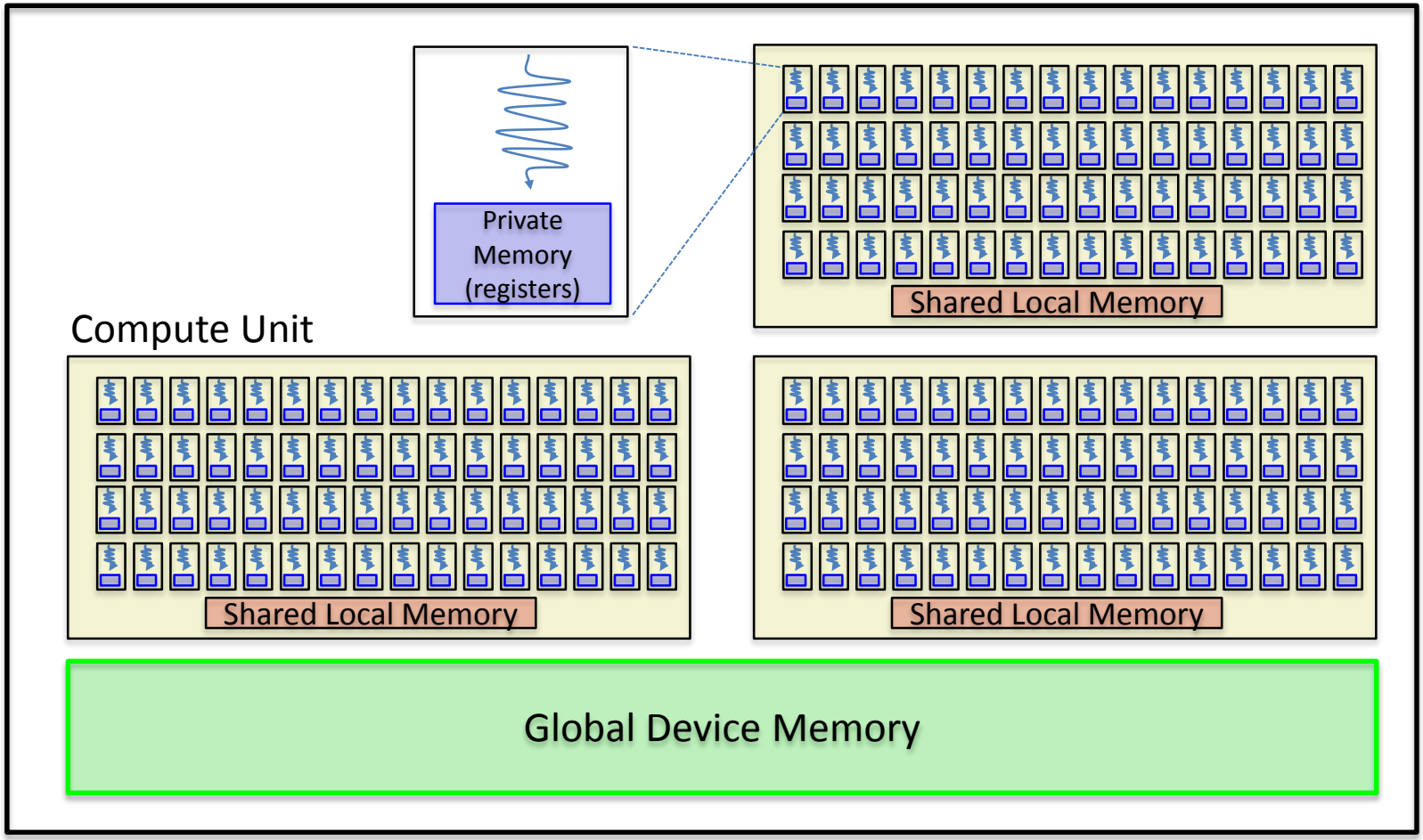
All 50 OpenCL™ kernels

AddOffsetKernel	AverageVelocitiesKernel	BatchSolveKernelContact	BatchSolveKernelFriction	ClearVelocitiesKernel	ContactToConstraintKernel
ContactToConstraintSplitKernel	CopyConstraintKernel	CountBodiesKernel	CreateBatches	CreateBatchesNew	FillFloatKernel
FillInt2Kernel	FillIntKernel	FillUnsignedIntKernel	LocalScanKernel	PrefixScanKernel	ReorderContactKernel
SearchSortDataLowerKernel	SearchSortDataUpperKernel	SetSortDataKernel	SolveContactJacobiKernel	SolveFrictionJacobiKernel	SortAndScatterKernel
SortAndScatterSortDataKernel	StreamCountKernel	StreamCountSortDataKernel	SubtractKernel	TopLevelScanKernel	UpdateBodyVelocitiesKernel
bvhTraversalKernel	clipCompoundsHullHullKernel	clipFacesAndContactReductionKernel	clipHullHullConcaveConvexKernel	clipHullHullKernel	computePairsKernel
computePairsKernelTwoArrays	copyAabbsKernel	copyTransformsToVBOKernel	extractManifoldAndAddContactKernel	findClippingFacesKernel	findCompoundPairsKernel
findConcaveSeparatingAxisKernel	findSeparatingAxisKernel	flipFloatKernel	initializeGpuAabbsFull	integrateTransformsKernel	newContactReductionKernel
processCompoundPairsKernel	scatterKernel				

Host and Device

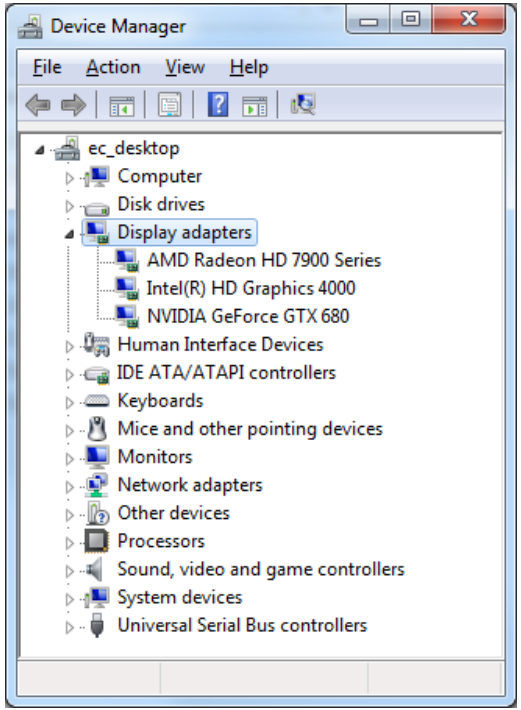


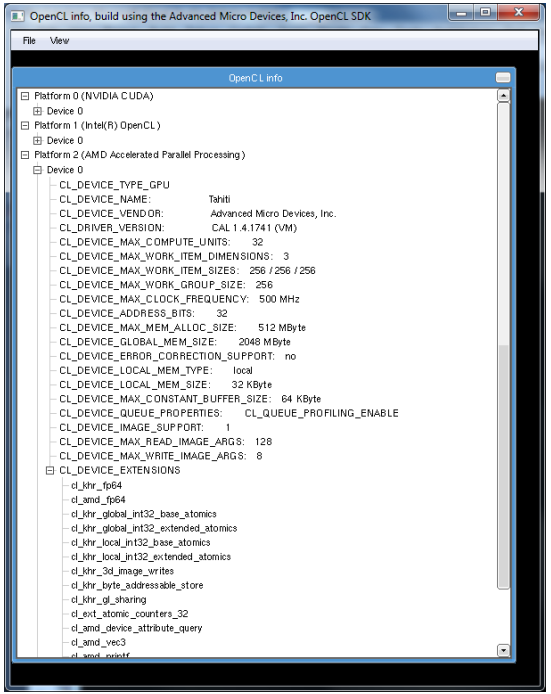
GPU in a nutshell



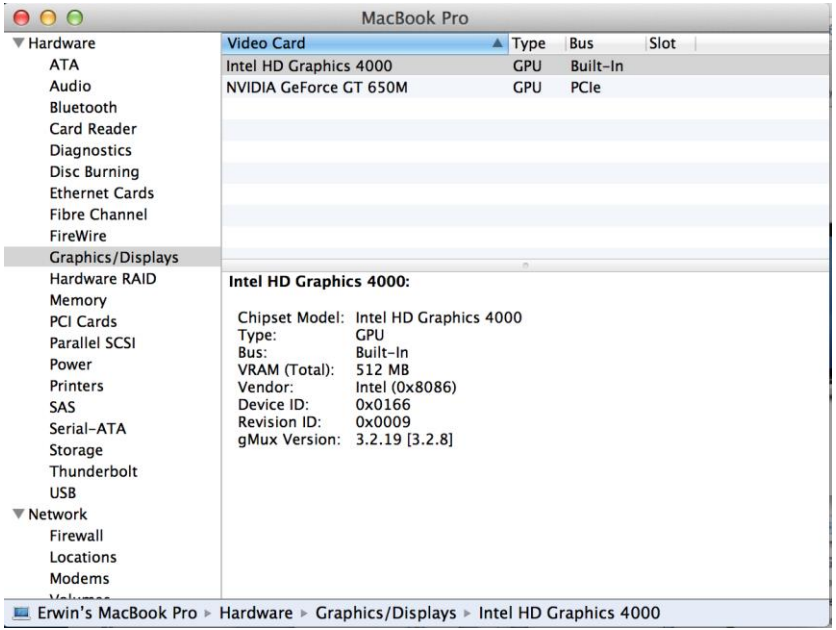
Windows GPU and CPU OpenCL Devices

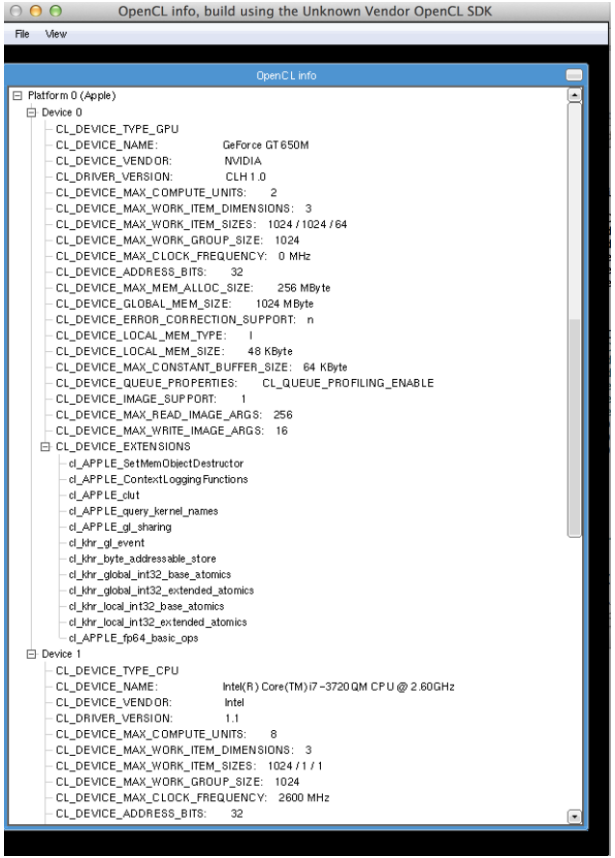
- Support for AMD Radeon, NVIDIA and Intel HD4000





Apple Mac OSX OpenCL Devices





Other GPGPU Devices

- Nexus 4 and 10 with ARM OpenCL SDK
- Apple iPad has a private OpenCL framework
- Sony Playstation 4 and other future game consoles

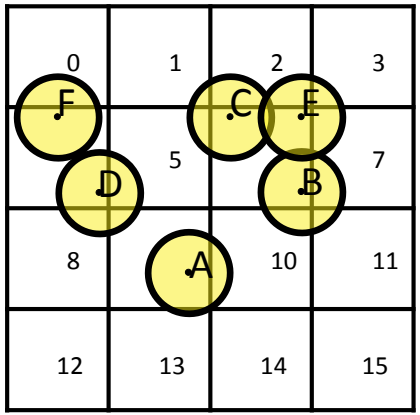
1st GPU rigid body pipeline (~2008-2010)

Detect Contact pairs

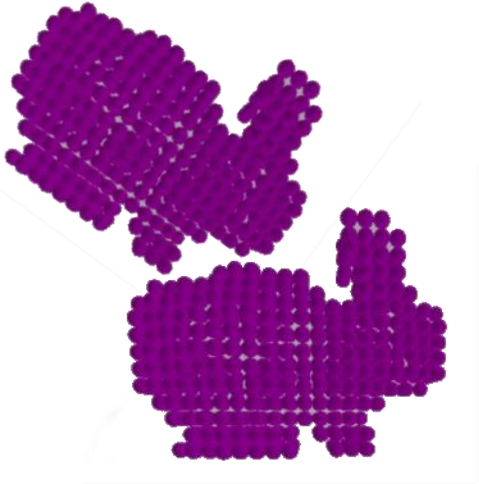
Compute contact points

Setup Contact constraints

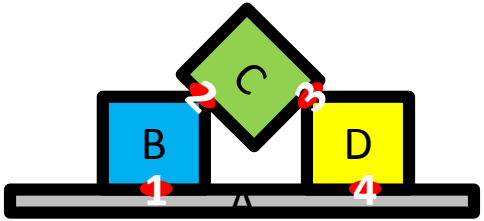
Solve constraints



Uniform grid



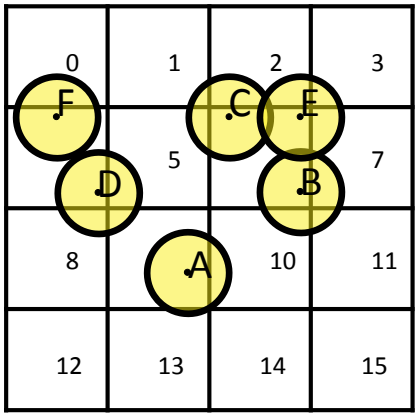
Spherical Voxelization



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

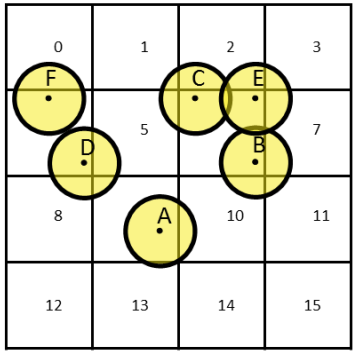
CPU batch and GPU solve (dispatched from CPU)

Uniform Grid



- Particle is also its own bounding volume (sphere)
- Each particle computes its cell index (hash)
- Each particle iterates over its own cell and neighborns

Uniform Grid and Parallel Primitives

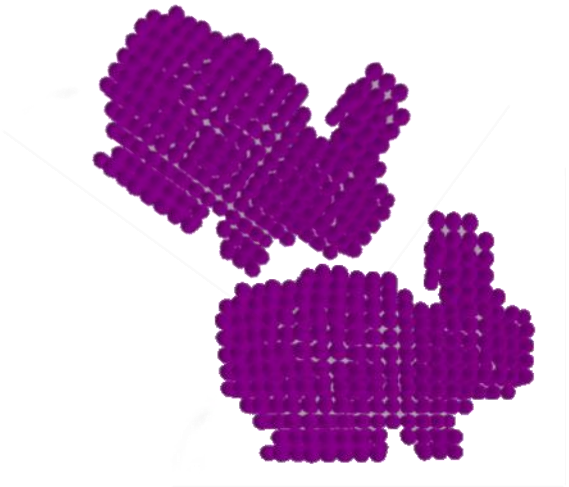


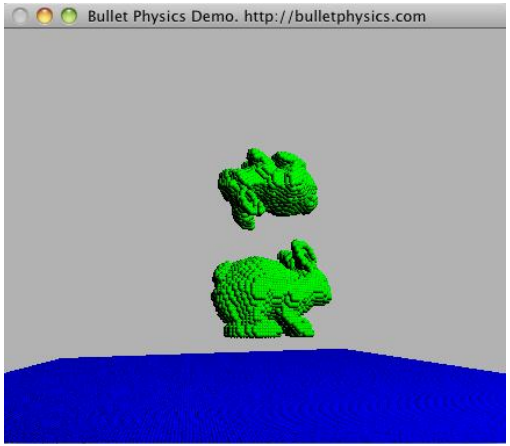
Cell Index	Cell Start
0	
1	
2	
3	
4	0
5	
6	2
7	
8	
9	5
10	
11	
12	
13	
14	
15	

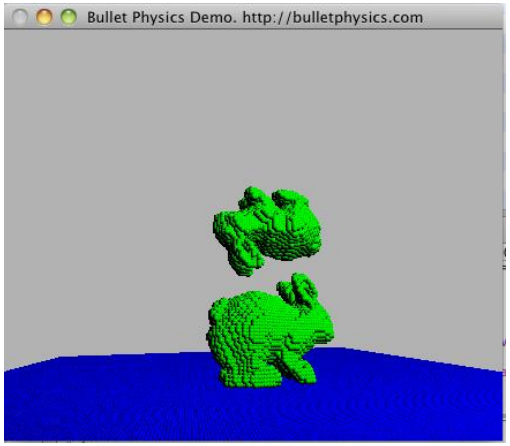
Array Index	Unsorted Cell ID, Particle ID	Sorted Cell ID Particle ID
0	9, A	4,D
1	6,B	4,F
2	6,C	6,B
3	4,D	6,C
4	6,E	6,E
5	4,F	9,A

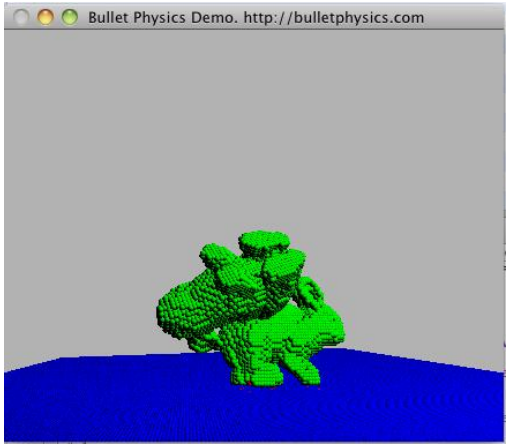
- Radix Sort the particles based on their cell index
- Use a prefix scan to compute the cell size and offset
- Fast OpenCL and DirectX11 Direct Compute implementation

Contact Generation

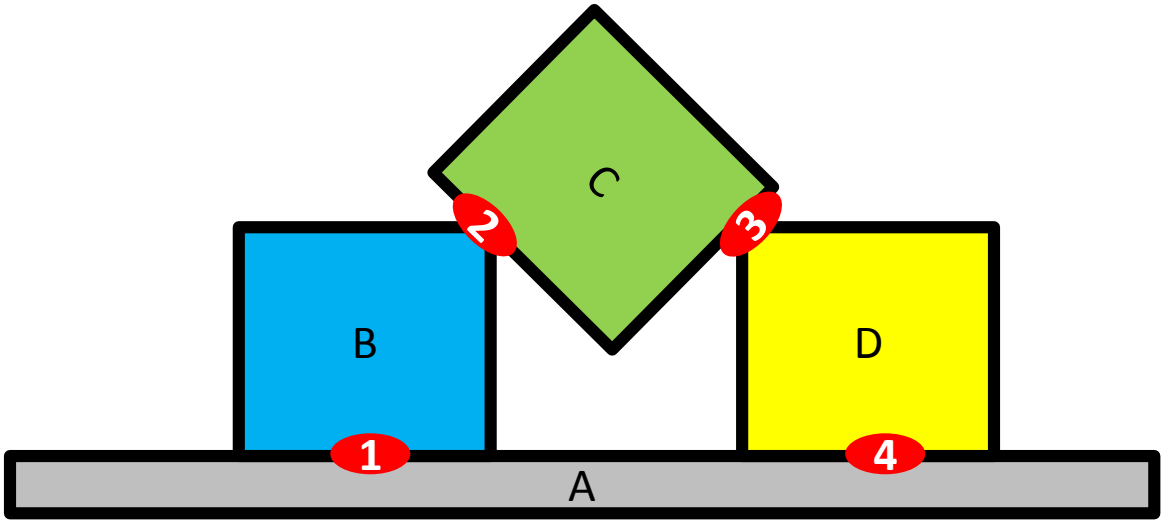



Bullet Physics Demo. http://bulletphysics.com

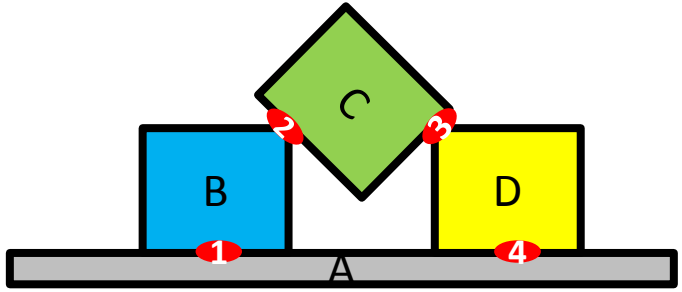

Bullet Physics Demo. http://bulletphysics.com


Bullet Physics Demo. http://bulletphysics.com

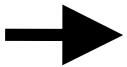
Constraint Generation



Reordering Constraints



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



	A	B	C	D
Batch 0	1	1	3	3
Batch 1	4	2	2	4

- Also known as Graph Coloring or Batching

CPU sequential batch creation

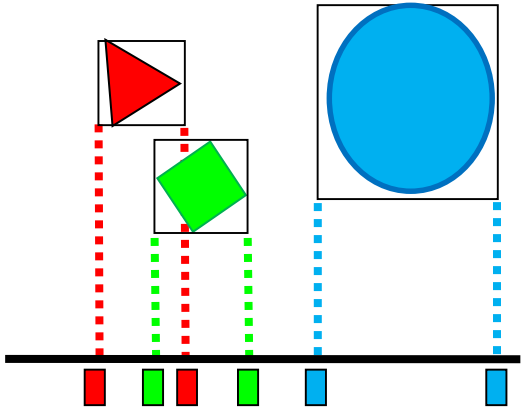
```
while( nIdxSrc ) {
    nIdxDst = 0;    int nCurrentBatch = 0;
    for(int i=0; i<N_FLG/32; i++) flg[i] = 0; //clear flag
    for(int i=0; i<nIdxSrc; i++)    {
        int idx = idxSrc[i];  btAssert( idx < n );
        //check if it can go
        int aIdx = cs[idx].m_bodyAPtr & FLG_MASK;   int bIdx = cs[idx].m_bodyBPtr & FLG_MASK;
        u32 aUnavailable = flg[ aIdx/32 ] & (1<<(aIdx&31));u32 bUnavailable = flg[ bIdx/32 ] & (1<<(bIdx&31));
        if( aUnavailable==0 && bUnavailable==0 )   {
            flg[ aIdx/32 ] |= (1<<(aIdx&31));   flg[ bIdx/32 ] |= (1<<(bIdx&31));
            cs[idx].getBatchIdx() = batchIdx;
            sortData[idx].m_key = batchIdx; sortData[idx].m_value = idx;
            nCurrentBatch++;
            if( nCurrentBatch == simdWidth ) {
               nCurrentBatch = 0;
               for(int i=0; i<N_FLG/32; i++) flg[i] = 0;
            }
        }
        else  {
            idxDst[nIdxDst++] = idx;
        }
    }
    swap2( idxSrc, idxDst ); swap2( nIdxSrc, nIdxDst );
    batchIdx ++;
}
```

Naïve GPU batch creation

- Use a single Compute Unit
- All threads in the Compute Unit synchronize the locking of bodies using atomics and barriers
- Didn’t scale well for larger scale simulations (>~30k)

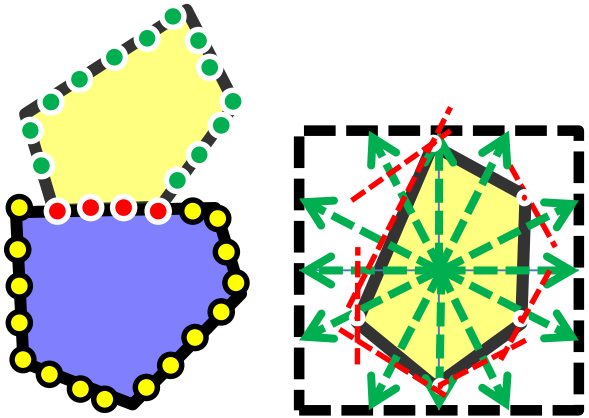
2nd GPU rigid body pipeline (~2010-2011)

Detect pairs



Mixed GPU/CPU broadphase and 1-axis parallel gSAP

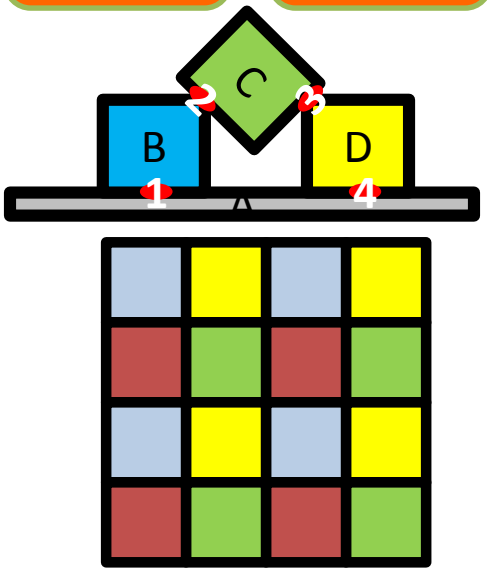
Compute contact points



Dual Surface/ Heightfield

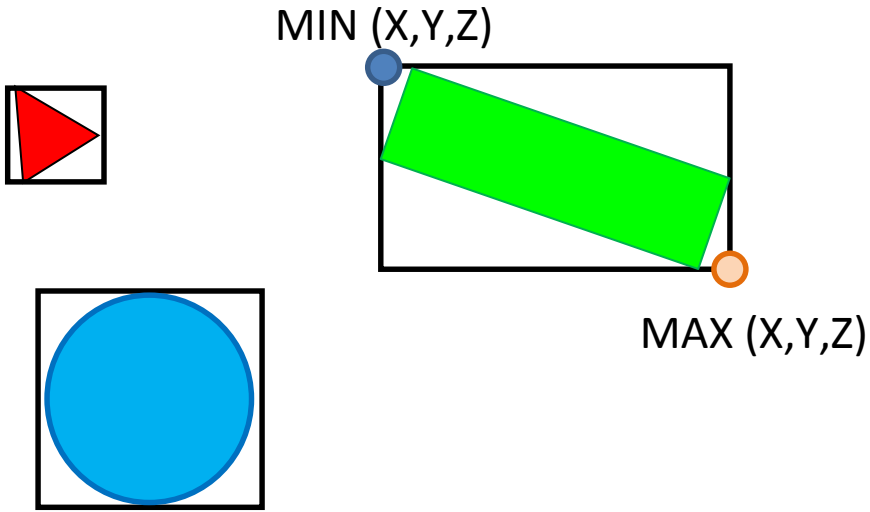
Setup constraints

Solve constraints



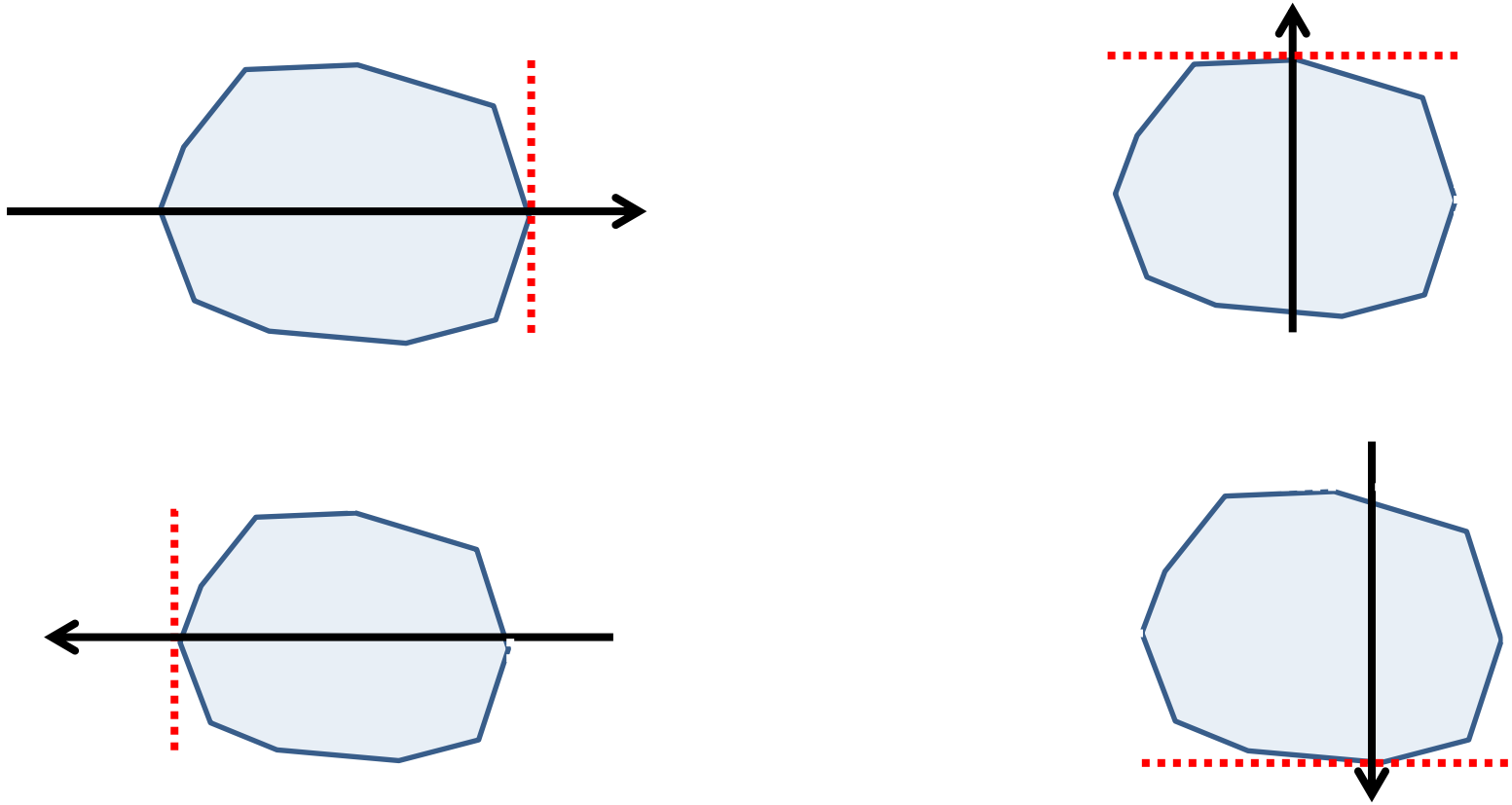
Dual Grid/ GPU batching & dispatch

Axis aligned bounding boxes (AABBs)



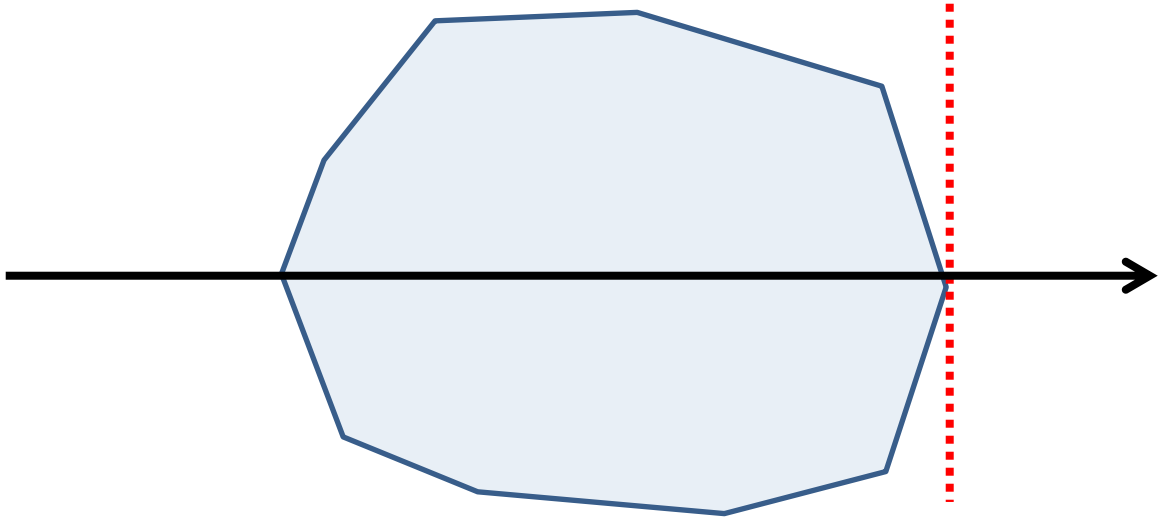
X min	X max
Y min	Y max
Z min	Z max
*	Object ID

Axis aligned bounding box



Support mapping

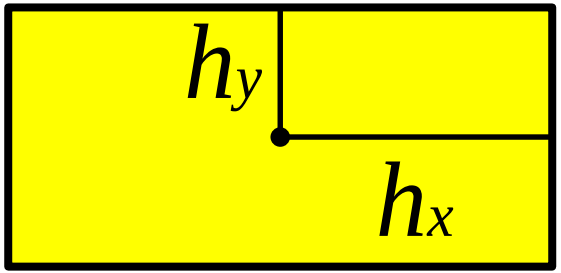
$$S_c(v) = \max\{v \cdot x : x \in C\}$$



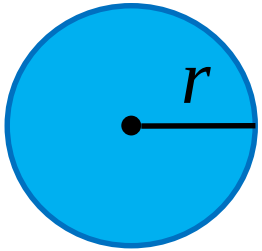
Support map for primitives

- Box with half extents h

$$S_{box}(v) = (sign(v_x)h_x, sign(v_y)h_y, sign(v_z)h_z)$$



$$S_{sphere}(v) = \frac{r}{|v|}v$$

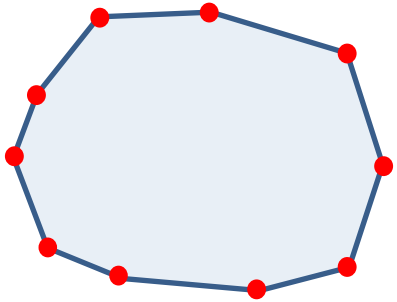


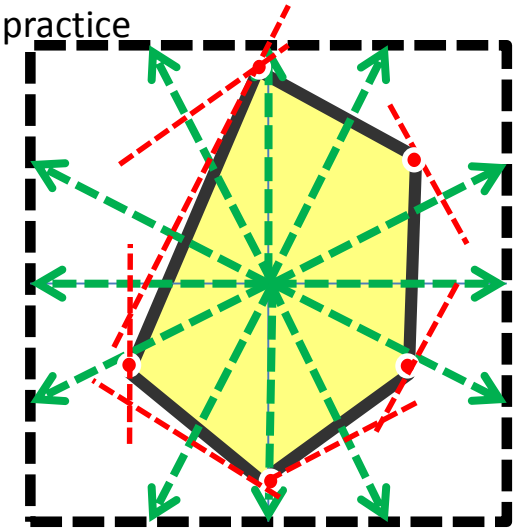
- Sphere with radius r

Support map for convex polyhedra

$$S_c(v) = \max\{v \cdot x : x \in C\}$$

- Brute force uniform operations (dot/max) on vertices are suitable for GPU
  - Outperforms Dobkin Kirkpatrick hierarchical optimization in practice



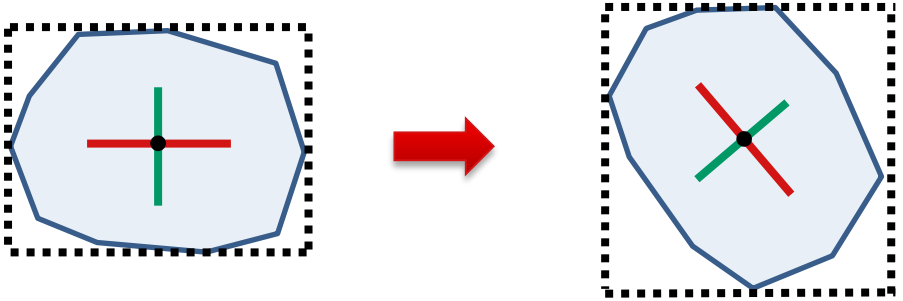


- Fast approximation using precomputed support cube map

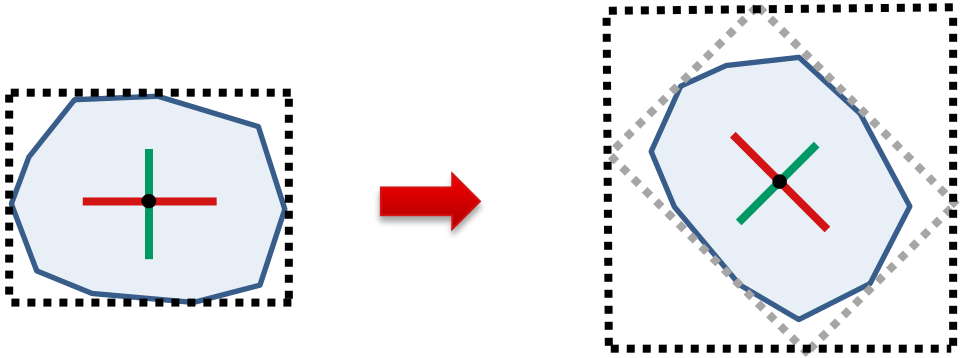
Worldspace AABB from Localspace AABB

- Affine transform

$$S_{Bx+c}(v) = B(S(B^t v)) + c$$



- Fast approximation using precomputed local aabb



- See opencl/gpu_rigidbody/kernels/updateAabbsKernel.cl

Host setup

```
int ciErrNum = 0;

int numObjects = fpio.m_numObjects;
int offset = fpio.m_positionOffset;

ciErrNum = clSetKernelArg(fpio.m_initializeGpuAabbsKernelFull, 0, sizeof(cl_mem), &bodies);
ciErrNum = clSetKernelArg(fpio.m_initializeGpuAabbsKernelFull, 1, sizeof(int), &numObjects);
ciErrNum = clSetKernelArg(fpio.m_initializeGpuAabbsKernelFull, 4, sizeof(cl_mem), (void*)&fpio.m_dlocalShapeAABB);
ciErrNum = clSetKernelArg(fpio.m_initializeGpuAabbsKernelFull, 5, sizeof(cl_mem), (void*)&fpio.m_dAABB);

size_t workGroupSize = 64;
size_t numWorkItems = workGroupSize*((numObjects+ (workGroupSize)) / workGroupSize);

ciErrNum = clEnqueueNDRangeKernel(fpio.m_cqCommandQue, fpio.m_initializeGpuAabbsKernel, 1, NULL, &numWorkItems,
&workGroupSize,0 ,0 ,0);
assert(ciErrNum==CL_SUCCESS);
```

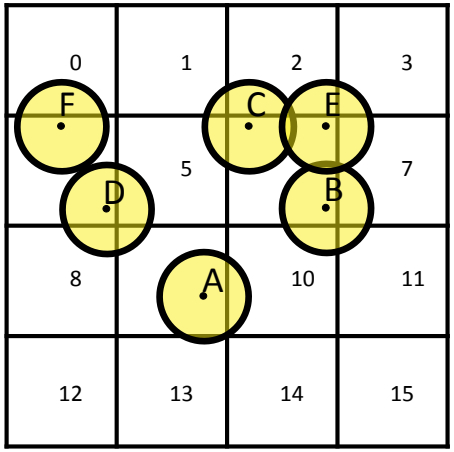
AABB OpenCL™ kernel

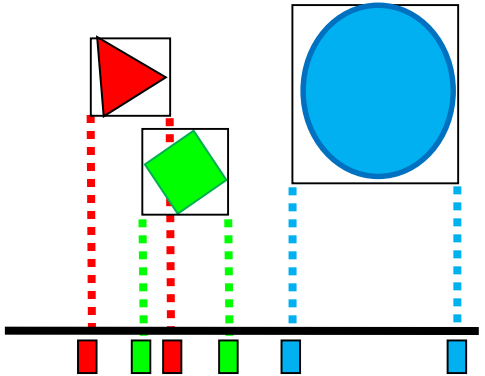
```
_void   initializeGpuAabbsFull(__global Body* gBodies, const int numNodes, __global btAABBCL* plocalShapeAABB,
__global btAABBCL* pWorldSpaceAABB)
{
      int nodeID = get_global_id(0);
      if( nodeID >= numNodes )
            return;
      float4 position = gBodies[nodeID].m_pos;
      float4 orientation = gBodies[nodeID].m_quat;
      int shapeIndex = gBodies[nodeID].m_shapeIdx;
      if (shapeIndex>=0)
      {
            btAABBCL minAabb = plocalShapeAABB[shapeIndex*2];
            btAABBCL maxAabb = plocalShapeAABB[shapeIndex*2+1];

            float4 halfExtents = ((float4)(maxAabb.fx - minAabb.fx,maxAabb.fy - minAabb.fy,maxAabb.fz -
            minAabb.fz,0.f))*0.5f;

            Matrix3x3 abs_b = qtGetRotationMatrix(orientation);
            float4 extent = (float4) (dot(abs_b.m_row[0],halfExtents),dot(abs_b.m_row[1],halfExtents),
                        dot(abs_b.m_row[2],halfExtents),0.f);
            pWorldSpaceAABB[nodeID*2] = position-extent;
            pWorldSpaceAABB[nodeID*2+1] = position+extent;
      }
}
```

See opencl/gpu_rigidbody/kernels/updateAabbsKernel.cl

Mixed CPU/GPU pair search

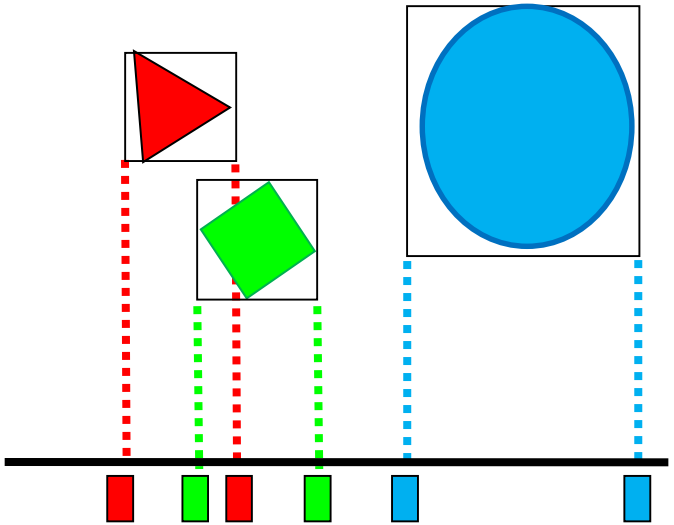
	Small	Large
Small	GPU	either
Large	either	CPU





Parallel 1-axis sort and sweep

- Find best sap axis
- Sort aabbs along this axis
- For each object, find and add overlapping pairs



- Works well with varying object sizes
- See also “Real-time Collision Culling of a Million Bodies on Graphics Processing Units” http://graphics.ewha.ac.kr/gSaP

GPU SAP OpenCL™ kernel optimizations

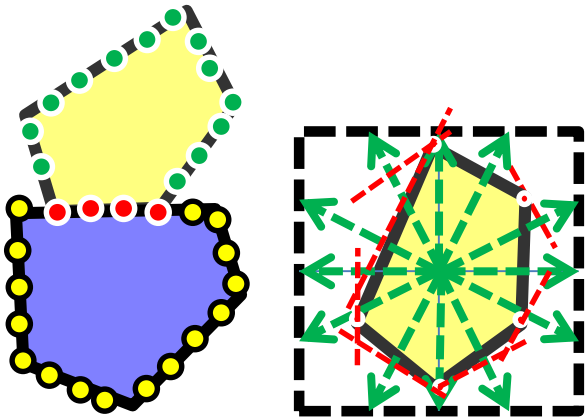
- Local memory
  - blocks to fetch AABBs and re-use them within a workgroup (requires a barrier)
- Reduce global atomic operations
  - Private memory to accumulate overlapping pairs (append buffer)
- Local atomics
  - Determine early exit condition for all work items within a workgroup
- Load balancing
  - One work item per object, multiple work items for large objects

- See opencl/gpu_broadphase/kernels/sapFast.cl and sap.cl

(contains un-optimized and optimized version of the kernel for comparison)

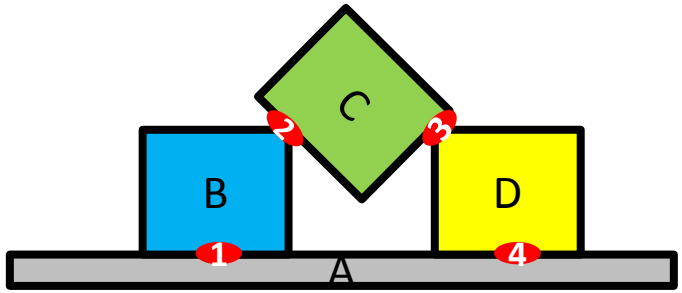
GPU Convex Heightfield contact generation

- Dual representation

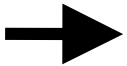


- SATHE, R. 2006. Collision detection shader using cube-maps. In ShaderX5, Charles River Media

Reordering Constraints Revisited

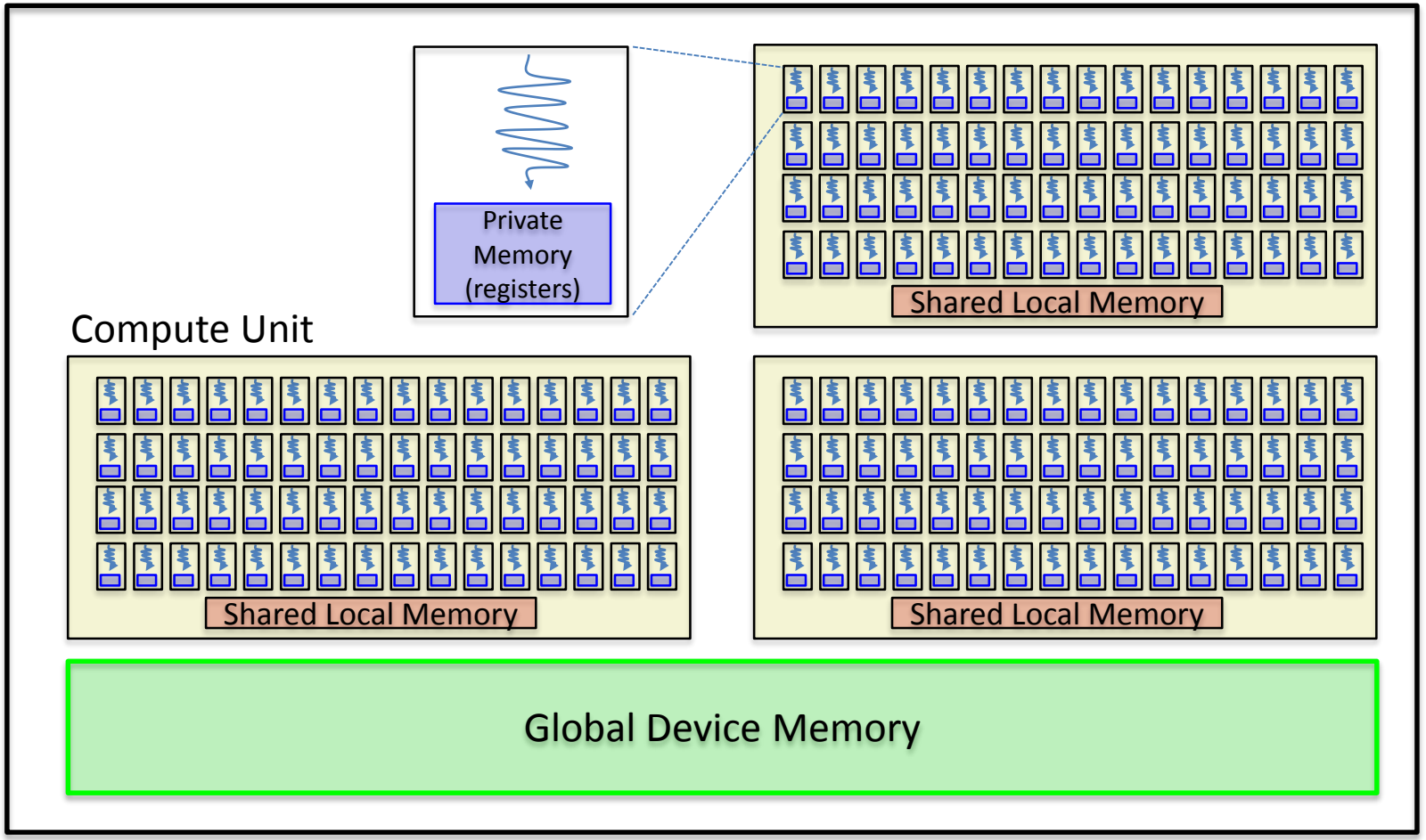


A	B	C	D
1	1		
	2	2	
		3	3
4			4

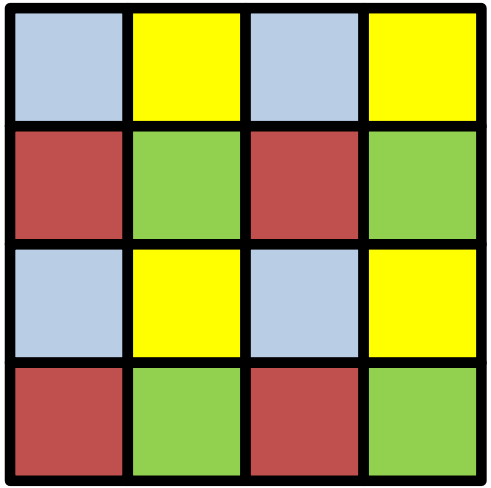


	A	B	C	D
Batch 0	1	1	3	3
Batch 1	4	2	2	4

Independent batch per Compute Unit?

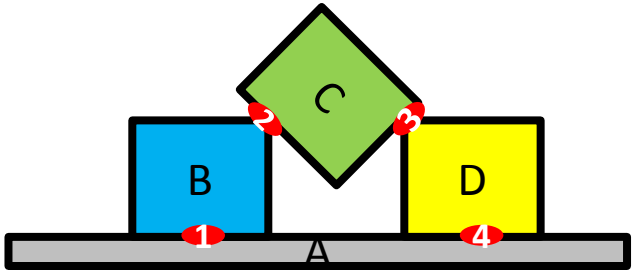


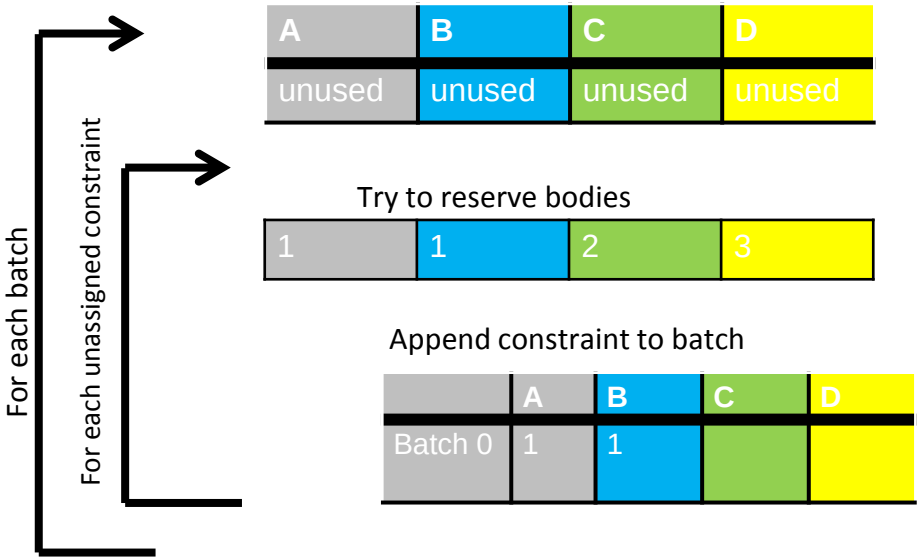
GPU parallel two stage batch creation



- Cell size > maximum dynamic object size
- Constraint are assigned to a cell
  - based on the center-of-mass location of the first active rigid body of the pair-wise constraint
- Non-neighboring cells can be processed in parallel

GPU iterative batching





- A SIMD can process the constraints in one cell
  - cannot be trivially parallelized by 64 threads in a SIMD
- Parallel threads in workgroup (same SIMD) use local atomics to lock rigid bodies
- Before locking attempt, first check if bodies are already used in previous iterations
- See “A parallel constraint solver for a rigid body simulation”, Takahiro Harada, http://dl.acm.org/citation.cfm?id=2077378.2077406

and opencl\gpu_rigidbody\kernels\batchingKernels.cl

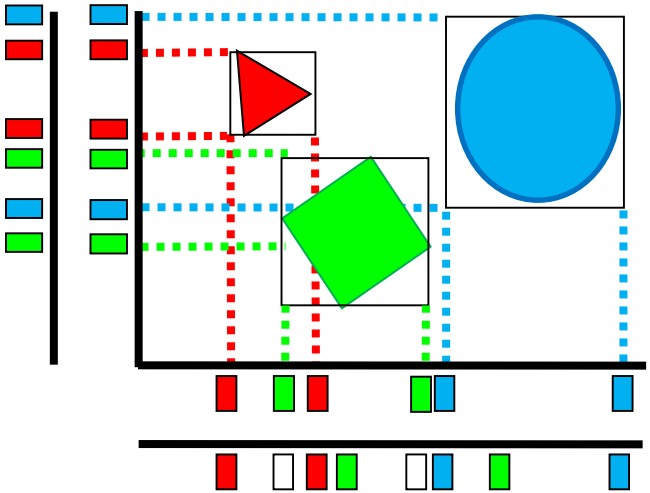
GPU parallel constraint solving

```
int idx=ldsStart+lIdx;
while (ldsCurBatch < maxBatch)   {
    for(; idx<end; )  {
        if (gConstraints[idx].m_batchIdx == ldsCurBatch)  {
            if( solveFriction )
                solveFrictionConstraint( gBodies, gShapes, &gConstraints[idx] );
            else
                solveContactConstraint( gBodies, gShapes, &gConstraints[idx] );
            idx+=64;
        } else {
            break;
        }
    }
    GROUP_LDS_BARRIER;
    if( lIdx == 0 )  {
        ldsCurBatch++;
    }
    GROUP_LDS_BARRIER;
}
```

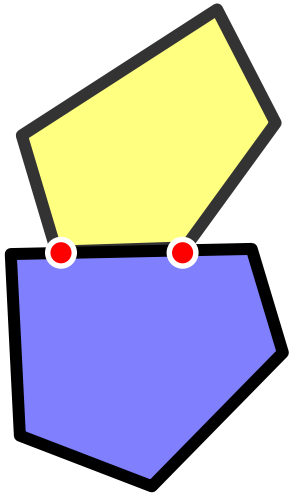
See “A parallel constraint solver for a rigid body simulation”, Takahiro Harada, http://dl.acm.org/citation.cfm?id=2077378.2077406
Source code at opencl\gpu_rigidbody\kernels\solveContact.cl and other solve*.cl

3rd GPU rigid body pipeline (2012-)

Detect pairs

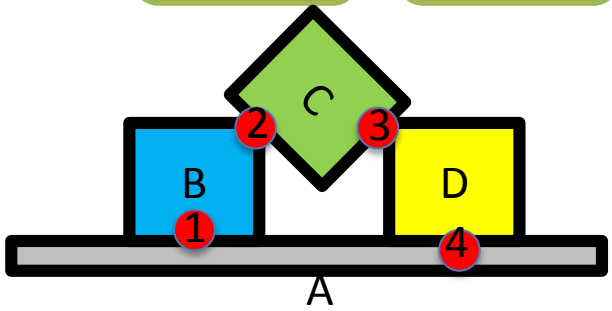


Compute contact points



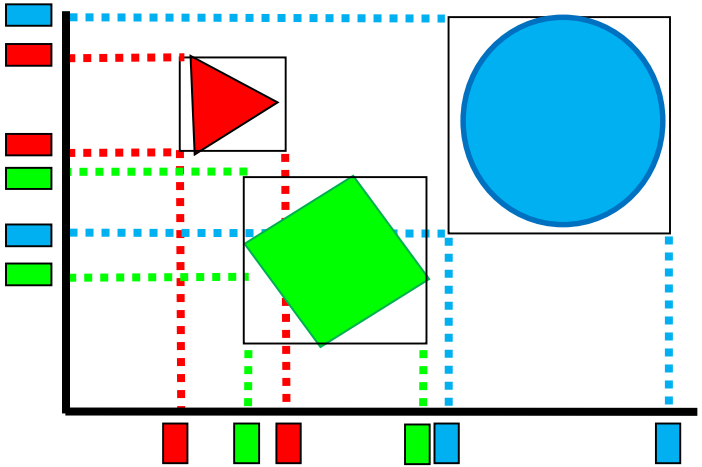
Setup constraints

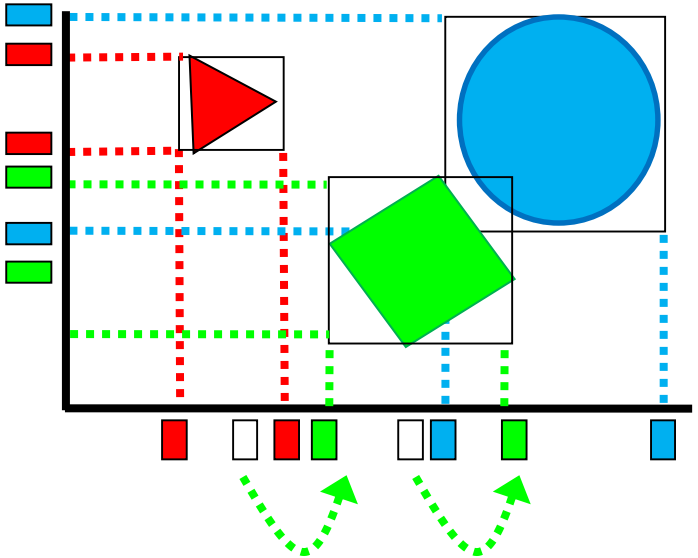
Solve constraints



A	B0	B1	C0	C1	D1	D1	A
1	1	2	2	3	3	4	4

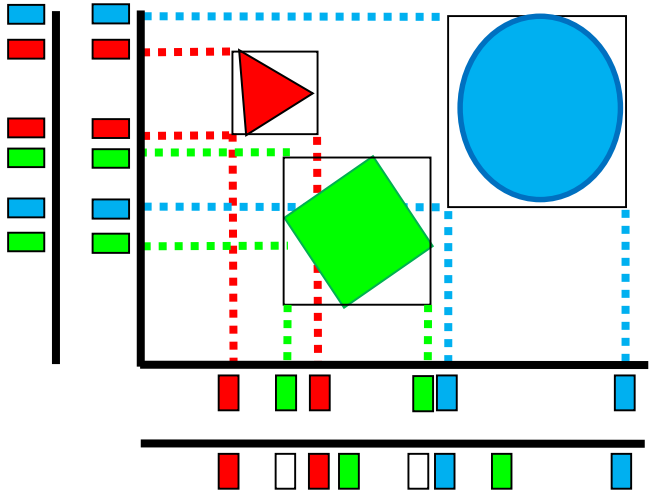
Sequential Incremental 3-axis SAP



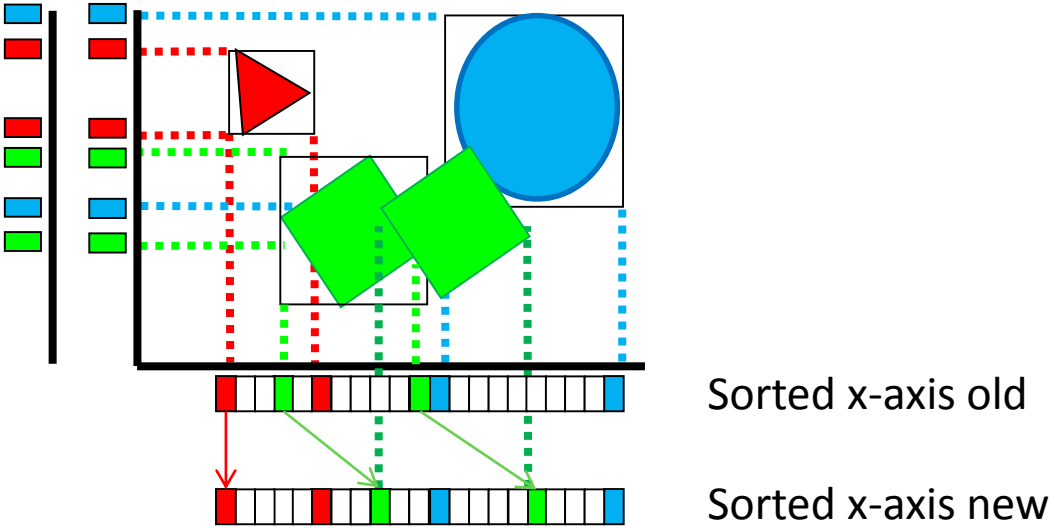


Parallel Incremental 3-axis SAP

- Parallel sort 3 axis
- Keep old and new sorted axis
  - 6 sorted axis in total



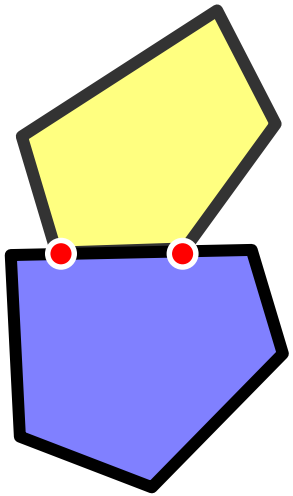
Parallel Incremental 3-axis SAP



- If begin or endpoint has same index do nothing
- Otherwise, range scan on old AND new axis
  - adding or removing pairs, similar to original SAP
- Read-only scan is embarrassingly parallel

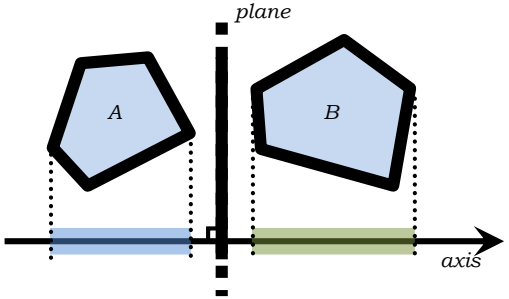
Convex versus convex collision

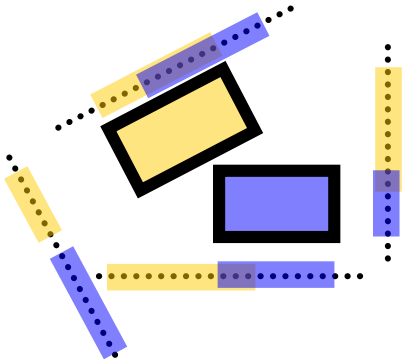
Compute contact points

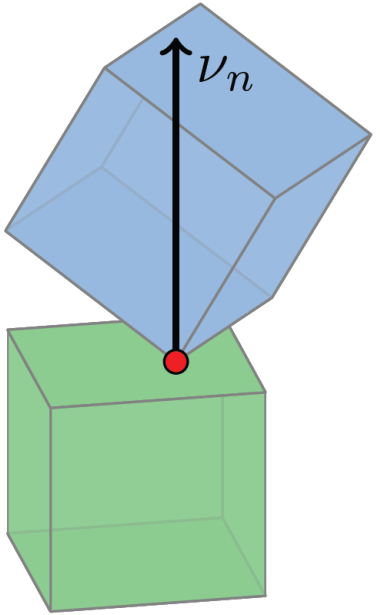


Separating axis test

- Face normal A
- Face normal B
- Edge-edge normal



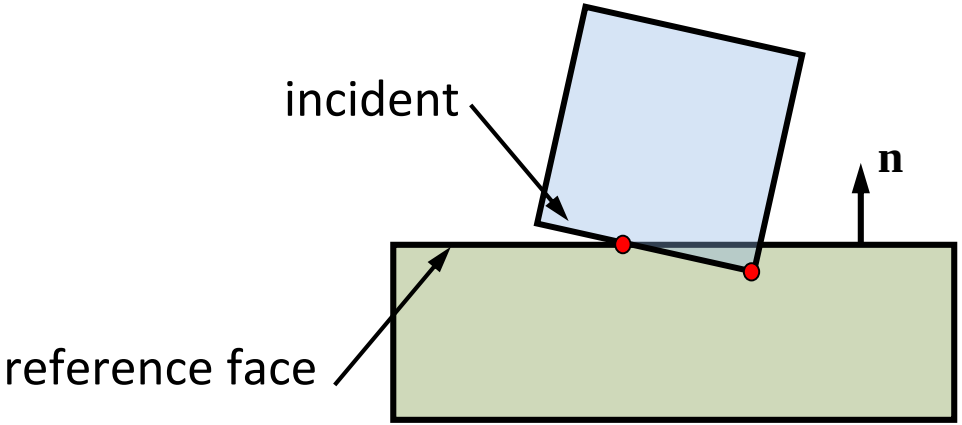


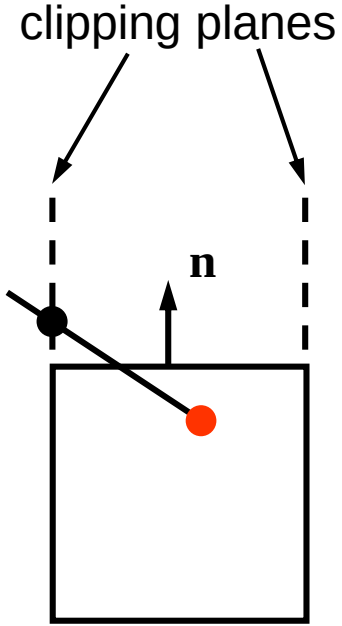


- Uniform work suits GPU very well: one work unit processes all SAT tests for o
- Precise solution and faster than height field approximation for low-resolution convex shapes
- See opencl/gpu_sat/kernels/sat.cl

Computing contact positions

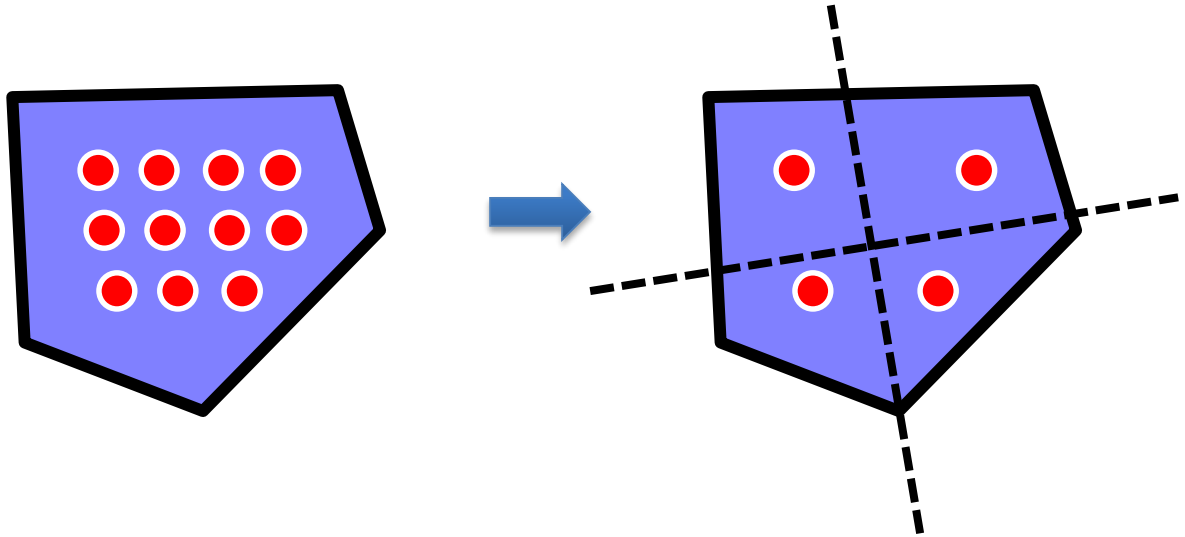
- Given the separating normal find incident face
- Clip incident face using Sutherland Hodgman clipping





- One work unit performs clipping for one pair, reduces contacts and appends to contact buffer
- See opencl/gpu_sat/kernels/satClipHullContacts.cl

GPU contact reduction



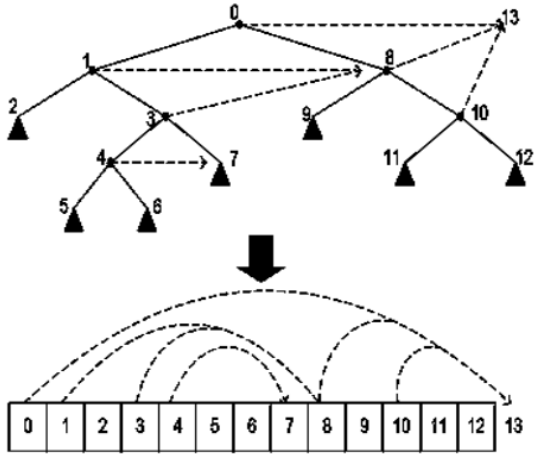
- See newContactReductionKernel in opencl/gpu_sat/kernels/satClipHullContacts.cl

SAT pipeline

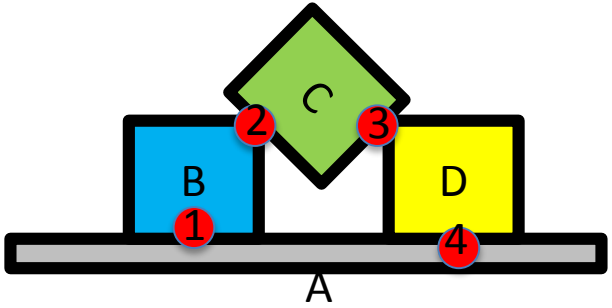
- Unified overlapping pairs
  - Broadphase Pairs
  - Compound Pairs
  - Concave triangle mesh pairs
- Break up more SAT stages to relief register pressure

GPU BVH traversal

- Create skip indices for faster traversal
- Create subtrees that fit in Local Memory
- Stream subtrees for entire wavefront/warp
- Quantize Nodes
  - 16 bytes/node

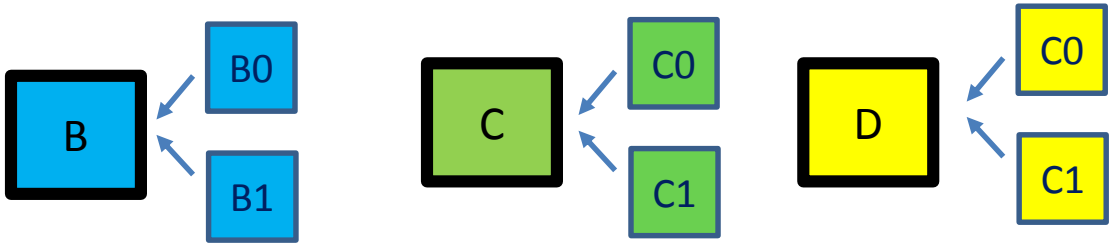


Mass Splitting+Jacobi = PGS



A	B0	B1	C0	C1	D1	D1	A
1	1	2	2	3	3	4	4

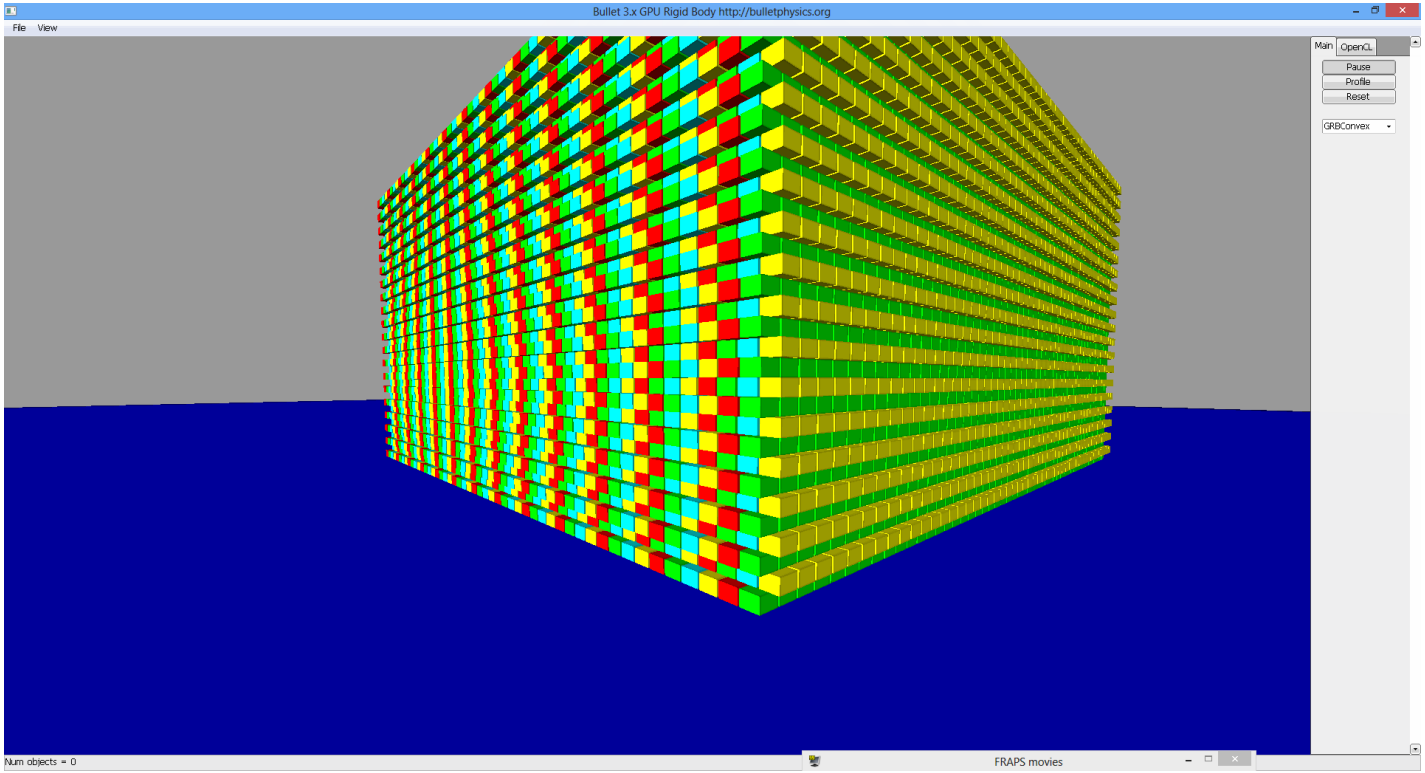
Parallel Jacobi



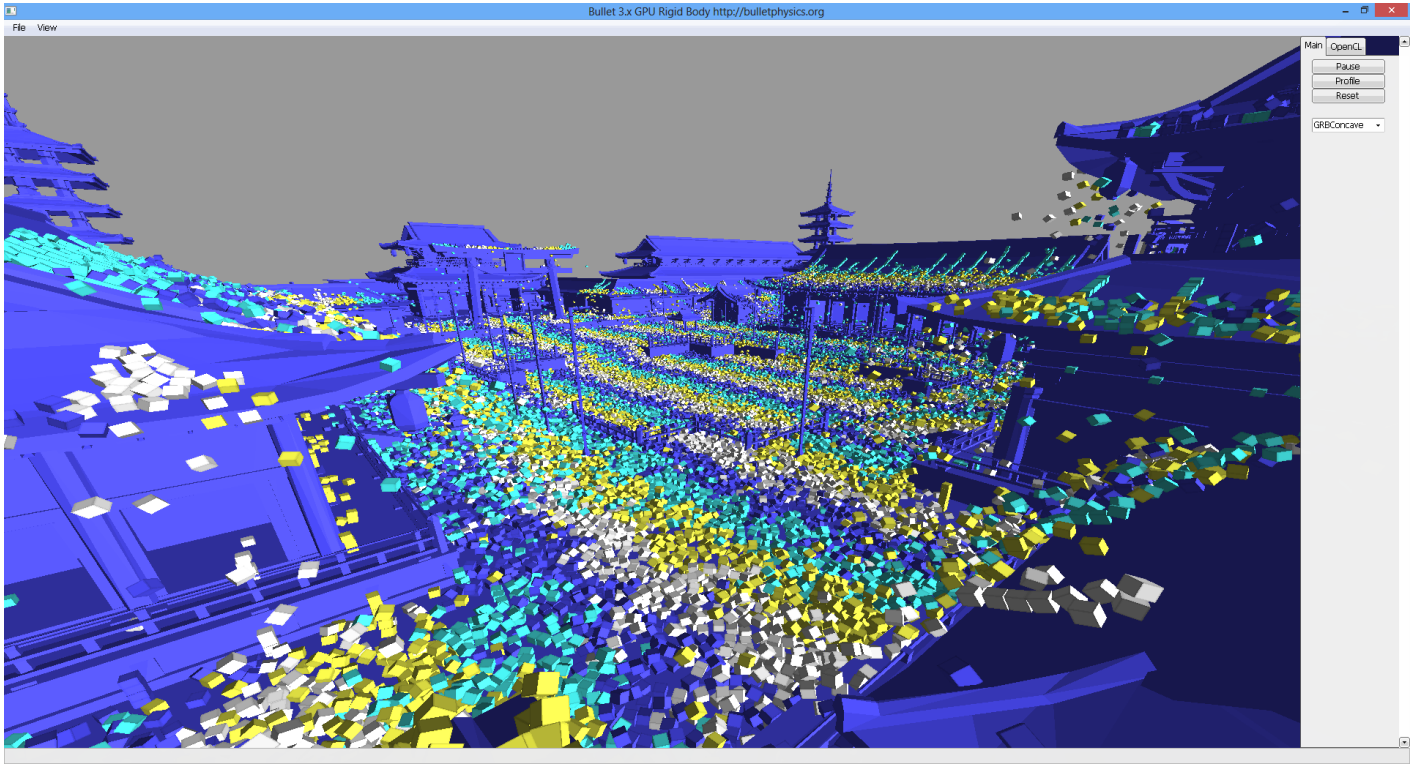
Averaging velocities

- See “Mass Splitting for Jitter-Free Parallel Rigid Body Simulation” by Tonge et. al.

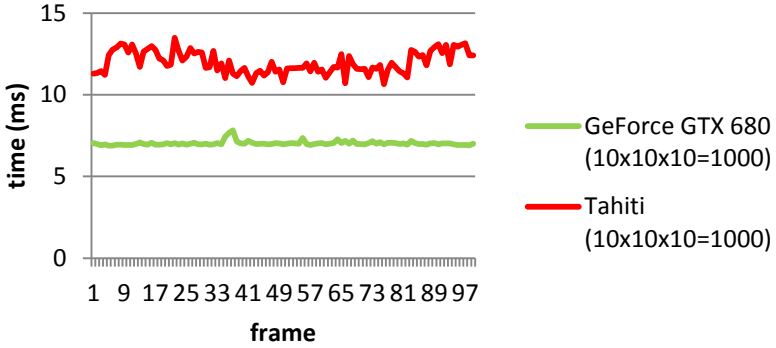
Test Scenario convex stack

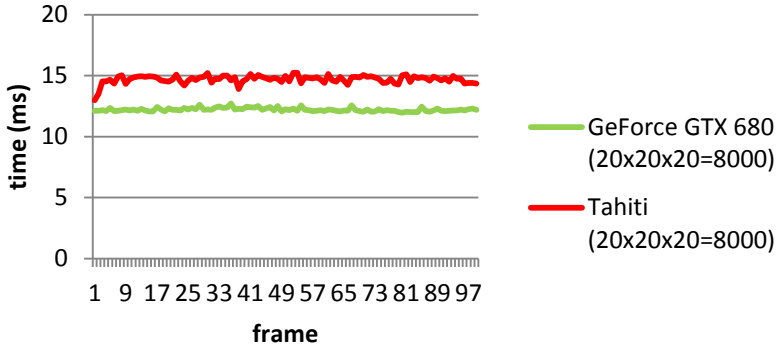


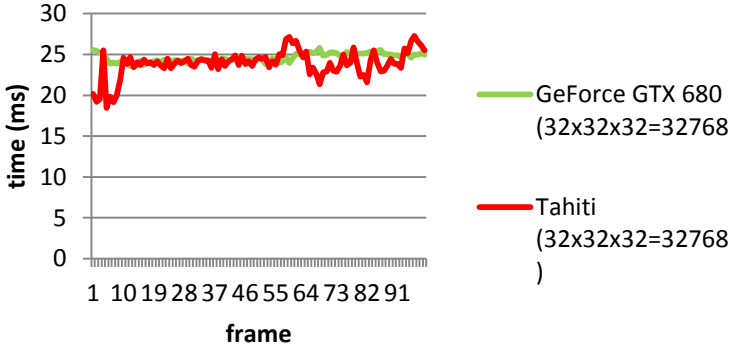
Test Scenario triangle mesh

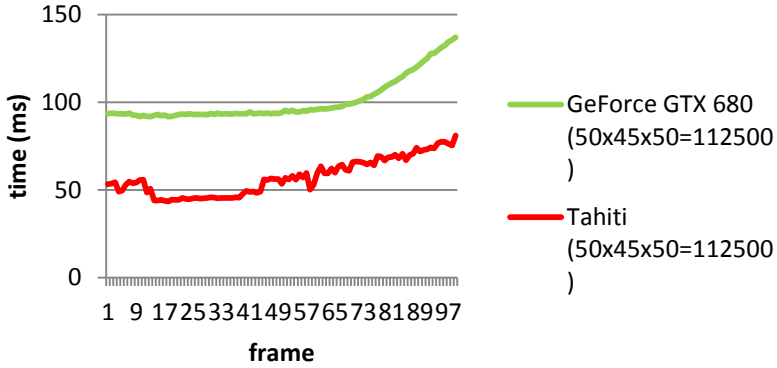


Performance









Timings for ½ million pairs (100k objects)

Profiling: stepSimulation (total running time: 73.233 ms) ---

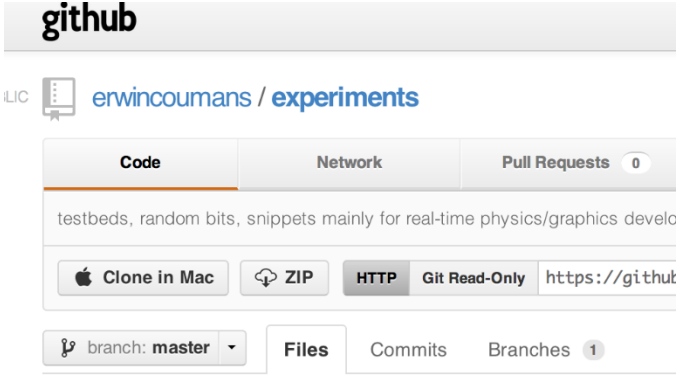
0 -- GPU solveContactConstraint (45.50 %) :: 33.319 ms / frame (1 calls)

1 -- batching (13.79 %) :: 10.099 ms / frame (1 calls)

2 -- computeConvexConvexContactsGPUSAT (15.62 %) :: 11.438 ms / frame (1 calls)

3 -- GPU SAP (23.60 %) :: 17.282 ms / frame (1 calls)

Build Instructions



All of the code discussed is open source

1. **Download ZIP or clone from**

http://github.com/erwincoumans/experiments

Windows Visual Studio

2. **Click** on build/vs2010.bat
3. **Open** build/vs2010/0MySolution.sln

Mac OSX Xcode or make

2. **Click** on build/xcode.command
3. **Open** build/xcode4/0MySolution.xcworkspace

Thank You!

- You can visit the forums at http://bulletphysics.org for further discussion or questions
- See previous slide for source code instructions