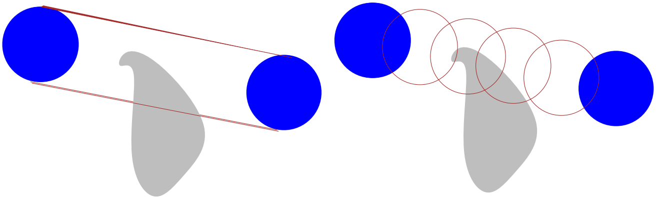
Collision Detection using Bounding Volume Hierarchies of K-DOPs
CPU vs CUDA

Davor Jovanoski

01.02.2011

Real-time Collision Detection

- Continuous - Priori - prediction
- Discrete - Posteriori -time-stamp
- Static environment - Flying objects
- Dynamic objects - dynamic environment

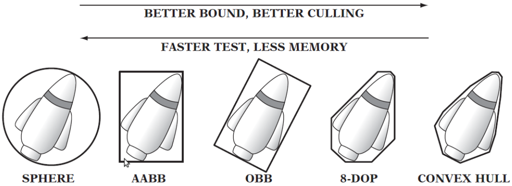


QUICKCD Algorithm [1]

- Bounding Volume Hierarchy
- Discrete Orientation Polytopes K-DOPs
- Flying Object - Rotation/Translation

Bounding Volumes

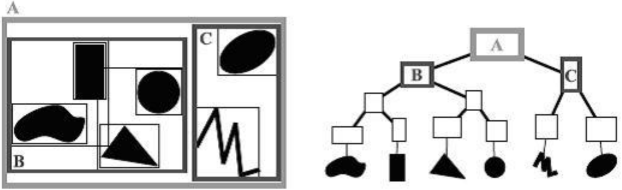
- Sphere
- AABB - Axis-aligned Bounding Boxes
- OBB - Oriented Bounding Boxes
- K-DOP - Discrete Orientation Polytopes
- Convex Hull



source: reference 4

BVH

- Set S of geometric objects-triangles, $BVH(S)$
- Each node $v \in BVH(S)$ corresponds to a subset, $S_v \subseteq S$
- Each intern-non-leaf has two or more children δ
- Each leaf corresponds to a singleton subset of S
- Each node is associated with a bounding volume $b(S_v)$
- Root is associated with the full set S



source: http://en.wikipedia.org/wiki/File:Example_of_bounding_volume_hierarchy.JPG

Desired BVH Characteristics

- The nodes contained in any given subtree should be near each other
- Each node in hierarchy should be of minimum volume
- The sum of all bounding volumes should be minimal
- Near root nodes should take priority
- Volume of overlap of sibling nodes should be minimal
- Balanced hierarchy
- Automatic generation - without preprocessing

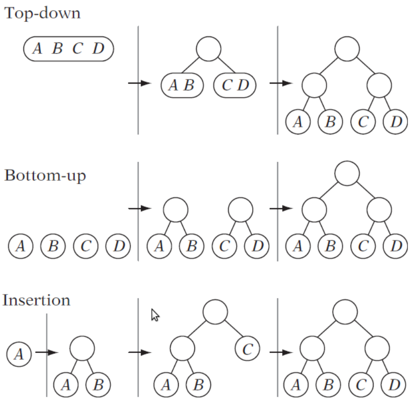
Degree of the Tree δ

- Maximum number of children that a node can have
- Height vs Degree
- Balanced Trees
- Amount of work of a query is proportional to $f(\delta) = (\delta - 1)\log_{\delta}(n)$
- Binary balanced trees

Top-Down vs Bottom-Up vs Insertion

- Top-Down is far more popular because of simplicity, with not so good trees
  - off-line - recursive, starts at the input set of primitives, which are than partitioned in δ subsets
- Bottom-Up is more complicated but better trees
  - off-line - slower construction, enclose each primitive within bounding volume, merge with criterion
- Insertion trees, insert one at a time
  - on-line - insert at the position that makes the tree grow as little as possible

Form



source: reference 4

Desired Partitions

- Minimize the sum of the volumes of the child volumes. The probability of an intersection between a bounding volume and the children is proportional to their volume.
- Minimize the maximum volume of the child volumes. Attempts to make the volumes more equal in size
- Minimize the volume of the intersection of the child volumes
- Divide primitives equally between the child volumes

Splitting rules

- Each node v corresponds to a set S_v together with a bounding volume $b(S_v)$
- Goal is to assign subsets of objects to each child v', of a node v in such a way as to minimize some function of the “sizes“/volumes of the children
- since there are binary trees $1/2(2^{S_v} - 2)$ different ways to do the splitting
- when to recalculate
  - all primitives fall on one side of the split plane
  - One or both child volumes end up with as many(nearly)primitives as the parent volume
  - Both child volumes are as large as the parent volume

Choice of Axis

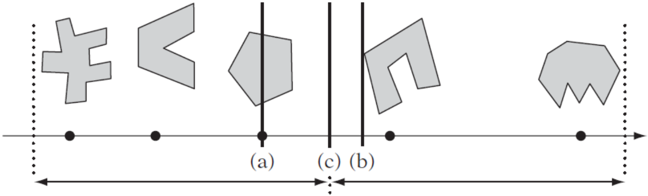
- Min Sum - choose the axis that minimizes the sum of the volumes of the two resulting children
- Min Max - Chose the axis that minimizes the larger of the volumes of the two resulting children
- Splatter - Project the centroids of the triangles onto each of the tree coordinate axes and calculate the variance of each of the resulting distributions
- Longest side- choose the axis along which the bounding volume is the longest
- Fixed axes that uses the bounding volume (k-dop normals)
- Axis through the two most distant points

Choice of Split Point

a Median of centroid coordinates $O(n)$

b Mean of the centroid coordinates $O(n)$

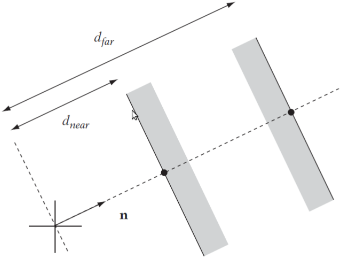
c Spatial median - two equal parts $O(c)$



source: reference 4

Slab

- A slab is the infinite region of space between two planes, defined by a normal *n* and two signed distances from the origin
- To form a closed 3D volume at least 3 slabs are required



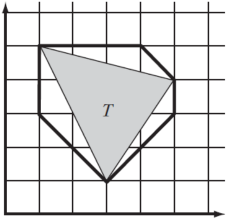
source: reference 4

Discrete orientation polytopes K-DOPs

- Based on the idea of slab-based volumes
- Almost identical to the slab based volumes, except the normals are fixed and shared amongst all k-dop bounding volumes
- It is not just any intersection of slabs, but the tightest slabs that form the body
- $(1,0,0)(0,1,0)(0,0,1)$- AABB 6-dops
- $(1,1,1)(1,-1,1)(1,1,-1)(1,-1,-1)$- Corners 8-dop
- $(1,1,0)(1,0,1),(0,1,1),(1,-1,0),(1,0,-1),(0,1,-1)$- Edges 12-dop
- 14-dop, 18-dop, 26-dop are derivative from the above

Compute K-DOP

- Dot product of the 3 vertices, with each of the normal vectors
- Find the minimum and maximum along the dot products
- The intersection of the set of slabs determines the k-dop

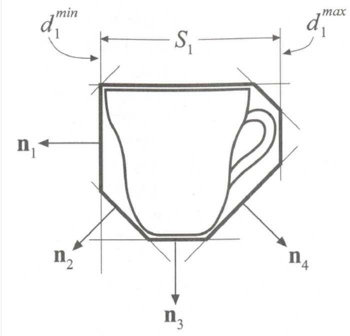


triangle $(3,1)(5,4)(1,5)$ with normals $(1,0)(0,1)(1,1)(1,-1)$

source: reference 4

Compute K-DOP

- K-DOP using slabs



source: reference 4

Compute K-DOP

- Sharing normals is computational advantage
- Storage is cheap, only min max from each normal must be stored
- Overlap testing, not more difficult than AABBs, simple check if any of the pair k/2 intervals do not overlap, the K-DOPs do not intersect
- Invariant on Translation
- Rotation leaves the volume unaligned with the predefined axes

K-DOP Rotation

- Must be realigned whenever the volume rotates
- Simple solution is to recompute the K-dop from scratch
- Hill-climbing algorithm - is an iterative algorithm that starts with an arbitrary solution, than attempts to find a better solution by incrementally changing single element of the solution
- Approximation method that tries to approximate k-dop

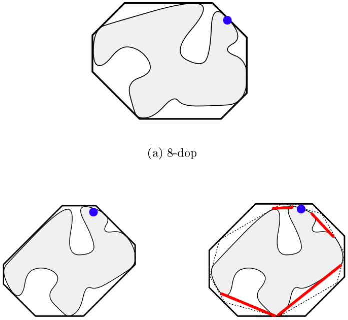
Hill-climbing

- B-rep - extension to the wire-frame method to represent the polygons composing an object surface is known as "boundary representation", because it describes the boundaries of the solid (in terms of an enclosing surface).
- B-rep of the convex hull of S_v in the previous position and orientation
- Check if a vertex that previously was extreme(maximal), is still maximal, in one of the k/2 directions
- If vertex no longer maximal then, local update "Climb the hill" by going to a neighboring vertex whose corresponding coordinate value increases the most
- exact, more expensive, used in near root level

Approximation

- Make a k-dop of the initial k-dop
- Due to number of rotations the k-dop calculation is non-sequential because the k-dop will become sphere-like
- stores only vertices $V(Sv)$
- need not to be the smallest k-dop bounding the object
- used in lower levels

Form



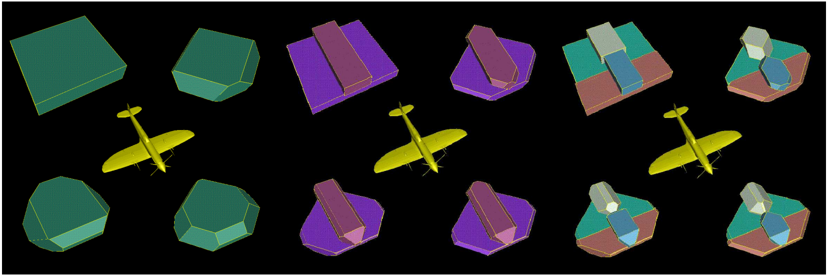
(a) 8-dop

(b) 8-dop of rotated object (c) 8-dop of rotated 8-dop

red: convex hull blue: extreme in vector (1,1)

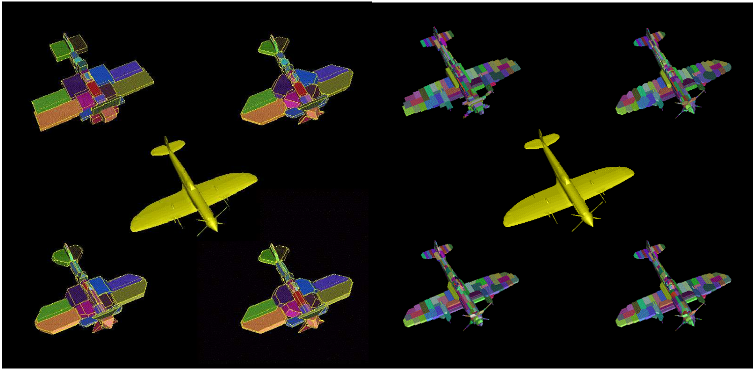
source: reference 1,2 with little changes

K-DOP Trees



Level 0,1,2
source: reference 1,2

K-DOP Trees



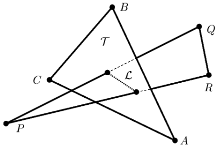
Level 5,8
source: reference 1,2

Collision Detection using BV-Trees and K-DOPs

- Build BVH for the environment
- Build BVH for the flying object
- Check intersections of the K-DOPs of both hierarchies
- If an intersection in Leaf found, use ERIT(Efficient and Reliable Intersection Tests)
- Tumble flying object

ERIT

1. Compute the plane equation of $\triangle ABC$. Exit with no intersection if the vertices of $\triangle PQR$ are all on the same side of the plane.
2. Compute the intersection between $\triangle PQR$ and the plane of $\triangle ABC$. This is a line segment in the plane of $\triangle ABC$.
3. Test if the line segment intersects or is contained in $\triangle PQR$. If so, the triangles intersect; otherwise they do not.



source: reference 3

Traverse - Algorithm

TraverseTrees(v_F,v_E)
  if $b(v_F) \cap b(v_E) \neq \emptyset$ then
    if v_E is a leaf then
      if v_F is a leaf then
        for each triangle t_E of S_{v_E}
          for each triangle t_F of S_{v_F}
            check test triangles t_E and t_F for intersections
      else for each child v_F of v_F
        TraverseTrees(v_F,v_E)
    else for each child v_E of v_E
      TraverseTrees(v_F,v_E)
return
Tumble(v_F)

Threads vs Blocks

- Threads3D;Blocks2D $\Upsilon := 1024; 2^{15}$
- Memory Host/Device/Texture/Constant/Shared
- Data vs Task parallelism
- Algorithms usually hybrid
- Everything can be programmed for CUDA, but will it be more efficient?

Reduction - Instructions Tree

1. Use each thread on one memory location, usually shared memory
2. Set margin at the middle of the memory
3. Let only the first half of the threads to compare/calculate the values with the second half of the memory after the margin
4. Synchronize threads, repeat the process until you get one value per Block
5. If synchronized properly, can calculate 2 different functions at the same time using left and right accordingly

Top-Down

- Splitting rules: Use task parallelism to check each of the algorithms
- Calculate centroids in one kernel call depending on Υ
- Using reduction calculate mean, variance(Splatter), min/max(longest side), median/sort on CPU of the centroids
- Store K-DOP normals in Constant memory for faster access
- Start different streams of Tasks for each of the axes variants
- Run kernel depending on Υ to calculate the dot-Product/slabs use reduction for max/min K-DOPs on each Level in each direction
- Compare on CPU the results of the spits the volumes of each of the splits
- repeat until threshold

Bottom-Up

- With start at max possible threshold given the set of primitives
- Calculate the centroids of all the primitives and all the k-dops
- Run a kernel that find the min volume of each two volumes, use the min/max values to calculate the k-dop of the father node
- Repeat step until we get the father volume, following balanced tree with min volume
- Using given threshold pre-calculations are neded for the defining of the subsets for the primitives

Discrete - Posteriori - Priori

- Calculate each time-stamp/Tumble in separate BVH for the flying object
- Preprocessing for all positions of the flying object
- Normals of the K-DOP are saved in Constant memory
- Each block calculates the Tumbling of the separate BVH
- Hill-climbing can use Υ threads to compare each extreme with the initial point*
- Split Υ threads for next level

Discrete - Posteriori - Priori-like

- Or each Tumble gets its own kernel call*
- Each block has Υ Threads, so each leaf can be checked with one thread
- Mark each leaf-intersection on all BVH trees, report back in Concurrent List
- Call another kernel that calculates all triangle intersections of the Concurrent List - dependent of the block/thread size Υ
- As result we get all collisions for each time-stamp
- Massive storage, useful for one flying object

Sequential

- Each time-stamp yields new kernel call for Tumble bottom-up
- Each K-DOP in lower levels updates the father with the min/max slabs
- One Concurrent BVH for the environment where locks are ON only on intersection checks
- With the use of Υ splits find the leaf intersections with the hierarchy and save them in Concurrent array
- Run kernel call to check all intersections of the primitives from the array
- Using more flying objects with each hierarchy leaf holding array of intersecting flying objects

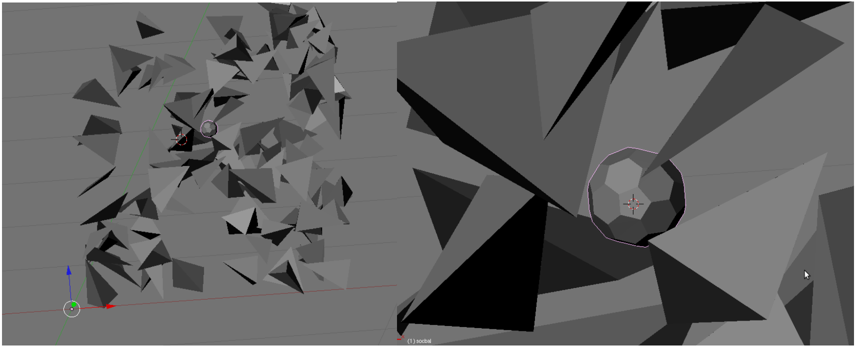
Subset in blocks

- Select special power of 2, depending on the Υ and height from the flying object tree
- Send each leaf to a specific cuda-block and check comparisons with the environment
- If there are more than one flying object, use the environment hirearchy to send it to the block
- Use a list inside the treenodes to reference the flying objects
- Check collisions with the environment, and each of the flying objects in the lists

Game Engines

source: http://udn.epicgames.com/Two/rsrc/Two/CollisionTutorial/show_collision.jpg

Socbal



References and Sources

1 J.T. Klosowski, M. Held, J.S.B. Mitchell, H. Sowizral, K. Zikan (1998); "Efficient Collision Detection Using Bounding Volume Hierarchies of k-DOPs"

2 J.T. Klosowski, 1998; "Efficient Collision Detection for Interactive 3D Graphics and Virtual Environments"

3 M. Held; "ERIT - A Collection of Efficient and Reliable Intersection Tests"

4 C. Ericson; "Real Time Collision Detection"

5 NVIDIA; "GPU Gems 3"

Thank You For Your Attention!!!