Collision Handling in Dynamic Simulation Environments

M. Teschner[1], B. Heidelberger[2], D. Manocha[3], N. Govindaraju[3], G. Zachmann[4], S. Kimmerle[5], J. Mezger[5], A. Fuhrmann[6]

[1] University of Freiburg, Germany
[2] ETH Zurich, Switzerland
[3] University of North Carolina at Chapel Hill, USA
[4] University of Bonn, Germany
[5] University of Tuebingen, Germany
[6] Fraunhofer Institute for Computer Graphics, Darmstadt, Germany

1. Introduction

In contrast to real-world objects, object representations in virtual environments have no notion of interpenetration. Therefore, algorithms for the detection of interfering object representations are an essential component in virtual environments. Applications are wide-spread and can be found in areas such as surgery simulation, games, cloth simulation, and virtual prototyping.

Early collision detection approaches have been presented in robotics and computational geometry more than twenty years ago. Nevertheless, collision detection is still a very active research topic in computer graphics. This ongoing interest is constantly documented by new results presented in journals and at major conferences, such as Siggraph and Eurographics. This interest in collision detection is based on

- recent advances in dynamic physically-based simulations which require efficient collision detection algorithms (see Fig. 1)
- new challenging problem domains such as deformable, time-critical, or continuous collision detection,
- advances in graphics hardware which is employed for image-space collision detection and for the acceleration of existing techniques.

In order to enable a realistic behavior of interacting objects in dynamic simulations, collision detection algorithms have to be accompanied by collision response schemes. These schemes process the collision information and compute a response with the objective of resolving the collision. For instance, distance field approaches provide the penetration depth of two objects which can easily be used for the collision response. However, other approaches provide less intuitive collision information, such as intersections of surface representations or certain patterns of the stencil buffer inside a GPU. Therefore, the nature of the information provided by a collision detection algorithm is an important characteristic in terms of its practicability.

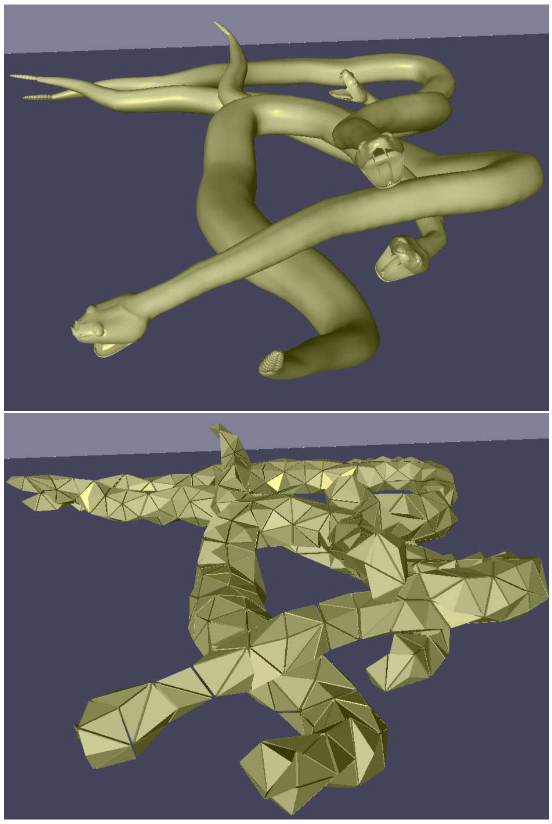


Figure 1: *Interactive environment with dynamically deforming objects and collision handling. Surface with high geometric complexity and the underlying tetrahedral mesh are shown.*

2. Summary

This tutorial will discuss collision detection algorithms with a special emphasis on the provided collision information. The potential combination with collision response schemes will be explained which is particular important for using collision detection algorithms in dynamic simulation environments. The tutorial will cover a large variety of relevant techniques.

The tutorial starts with basic concepts, such as bounding-volume hierarchies, spatial partitioning, distance fields, and proximity queries. The idea of image-space collision detection is derived as a special case of spatial partitioning and it is illustrated how graphics hardware can be used to accelerate these methods. Based on the provided collision information, the potential combination with collision response schemes will be discussed for all techniques.

The tutorial proceeds with further collision detection challenges that are particular important for dynamic simulation environments. Approaches to self-collision detection, as they can occur in deformable modeling, will be discussed. Stochastic methods, that can be used for time-critical collision detection, will be explained. Further, continuous collision detection will be introduced which aims at solving problems related to discrete-time simulations.

3. Proposed Length

- full-day tutorial

4. Topics

- Bounding-Volume Hierarchies
- Spatial Partitioning
- Distance Fields
- Proximity Queries
- Image-Space Collision Detection
- Detection of Self-Collisions
- Stochastic Methods
- Continuous Collision Detection

5. Tutorial Syllabus

Basic Techniques (half day). In this part of the tutorial, four main concepts of collision detection algorithms will be explained: bounding-volume hierarchies, spatial partitioning, distance fields, and proximity queries. Advantages, drawbacks, and relevance of the collision information with respect to the considered application in simulation environments will be discussed.

Advanced Techniques (half day). The main topic in this part is image-space collision detection. A variety of recent approaches will be explained and discussed. Further, solutions to specific collision detection problems inherent to dynamic simulation environments will be discussed, namely self-collisions, time-critical collision detection, and continuous collision detection.

6. Suggestions for Shorter Presentations

In the case of a condensed half-day tutorial, the presentations would be focused on recent advances in collision handling, such as GPU-accelerated image-space collision detection, stochastic methods for time-critical collision detection, challenges in continuous collision detection, and approximate proximity queries for consistent collision response.

7. Prerequisites

The participants should have a working knowledge of spatial data structures, graphics hardware, and dynamic simulation environments.

8. Organizer

Prof. Dr.-Ing. Matthias Teschner

Computer Graphics Laboratory
Computer Science Department
Albert-Ludwigs-University Freiburg

Georges-Koehler-Allee 052
79110 Freiburg im Breisgau
Germany

phone +49 761 203 8281
fax +49 761 203 8262
mail teschner@informatik.uni-freiburg.de
http http://cg.informatik.uni-freiburg.de/

9. Speakers

Matthias Teschner received the PhD degree in Electrical Engineering from the University of Erlangen-Nuremberg in 2000. From 2001 to 2004, he was research associate at Stanford University and at the ETH Zurich. Currently, he is professor of Computer Science and head of the Computer Graphics Laboratory at the University of Freiburg. His reserach interests comprise real-time rendering, scientific computing, physical simulation, computer animation, computational geometry, collision handling, and human perception of motion. His research is particularly focused on real-time physically-based modeling of interacting deformable objects and fluids with applications in entertainment technology and medical simulation. Matthias Teschner has contributed to the field of physically-based modeling and collision handling in several papers. At Eurographics 2004, he organized a State-of-the-Art report on collision detection. At IEEE VR 2005, he will participate in a tutorial on collision detection.

Bruno Heidelberger received his MSc degree in Computer Science from the Swiss Federal Institute of Technology, Zurich, Switzerland in 2002. He is currently pursuing his PhD as a member of the Computer Graphics Laboratory at ETH Zurich. His research interests are real-time computer graphics, especially collision detection, collision response and deformable modeling. He has published numerous papers at international conferences in the aforementioned research areas and contributed to the State-of-the-Art Report on "Collision Detection for Deformable Objects" at Eurographics 2004.

Dinesh Manocha is currently a professor of Computer Science at the University of North Carolina at Chapel Hill. He received his B.Tech. degree in Computer Science and Engineering from the Indian Institute of Technology, Delhi in 1987; M.S. and Ph.D. in Computer Science at the University of California at Berkeley in 1990 and 1992, respectively. He received Alfred and Chella D. Moore fellowship and IBM graduate fellowship in 1988 and 1991, respectively, and a Junior Faculty Award in 1992. He was selected an Alfred P. Sloan Research Fellow, received NSF Career Award in 1995 and Office of Naval Research Young Investigator Award in 1996, Honda Research Initiation Award in 1997, and Hettleman Prize for scholarly achievement at UNC Chapel Hill in 1998. He has also received best paper awards at the ACM SuperComputing, ACM Multimedia and Eurographics conferences. His research interests include geometric and solid modeling, interactive computer graphics, physically-based modeling, virtual environments, robotics and scientific computation. His research has been sponsored by ARO, DARPA, DOE, Honda, Intel, NSF, ONR and Sloan Foundation. He has published more than 120 papers in leading conferences and journals on computer graphics, geometric and solid modeling, robotics, symbolic and numeric computation, virtual reality, molecular modeling and computational geometry. He has served as a program committee member for many leading conferences on virtual reality, computer graphics, computational geometry, geometric and solid modeling, animation and molecular modeling. He was the program co-chair for the first ACM Siggraph workshop on simulation and interaction in virtual environments and program chair of first ACM Workshop on Applied Computational Geometry. He was the guest co-editor of special issues of International Journal of Computational Geometry and Applications. He is a member of the editorial boards of IEEE Transactions on Visualization and Computer Graphics, and Graphical Models and Imaging Processing.

Naga Govindaraju is currently research assistant professor of Computer Science at the University of North Carolina at Chapel Hill. He received his B.Tech. degree in Computer Science and Engineering from the Indian Institute of Technology, Bombay in 2001, M.S. and Ph.D. in Computer Science at the University of North Carolina at Chapel Hill in 2003 and 2004, respectively. His research interests include computer graphics, computational geometry, data bases, data mining, graphics hardware, parallel and distributed computing. He serves as a program committee member for the Pacific Graphics 2005. Naga Govindaraju has contributed to the field of GPU-accelerated collision detection in several papers, and tutorials. At Siggraph 2004, he was co-presenter of a course on general purpose computation on graphics hardware.

Gabriel Zachmann is professor for computer graphics at Clasuthal University since 2005. Prior to that, he was assistant professor with the computer graphics group at Bonn University. He received a PhD in computer science from Darmstadt University in 2000. From 1994 until 2001, he was with the virtual reality group at the Fraunhofer Institute for Computer Graphics in Darmstadt, where he carried out many industrial projects in the area of virtual prototyping. Zachmann has published many papers at international conferences in areas like collision detection, virtual prototyping, intuitive interaction, mesh processing, and camera-based hand tracking. He has also served on various international program committees.

Stefan Kimmerle studied Physics and Chemistry in Tuebingen and San Diego. In 2000, he received his Diploma in Physics from the University of Tuebingen. Since 2001, he is a PhD student at the graphics research group at GRIS. In 2003 and 2004, he was an invited researcher at GRAVIR, INRIA Rhone-Alpes in Grenoble. His main research interests are physically-based modeling and collision detection for deformable objects. His special interest is the simulation of virtual cloth. Stefan Kimmerle has contributed to the field of collision detection and cloth simulation in several papers, State-of-the-Art reports and tutorials. At Eurographics 2004, he was co-presenter of a tutorial on the real-time simulation of cloth and of a State-of-the-Art report on collision detection of deformable objects.

Johannes Mezger received his Diploma in Computer Science from the University of Tuebingen, Germany, in 2002. Since then he is PhD student and research associate at the graphics research group GRIS in Tuebingen. His research interests include collision detection and the simulation of deforming objects. Johannes Mezger has contributed to the field of collision detection and cloth simulation in several publications.

Arnulph Fuhrmann studied Computer Science at the University of Technology in Darmstadt and received his Diploma in 2001. Since 2001, he is a member of the Animation and Image Communication research group at the Fraunhofer Institute for Computer Graphics. His main research interests are physically based modeling, animation of clothes and collision detection for deformable objects. In area of collision detection, he has published many papers at international conferences. He has contributed to a State-of-the-Art report on collision detection at Eurographics 2004.

10. Course Notes Description

This tutorial builds on lecture material from the University of Freiburg, ETH Zurich, University of North Carolina at

Chapel Hill, and the University of Bonn. Further, material from a previous STAR presentation at Eurographics 2004, a tutorial at IEEE VR 2005, and a course at Siggraph 2004 will be used. Since all presenters actively contribute to the area of collision detection, all presentations will be accompanied by videos and software demonstrations.

Further course notes and illustrating videos can be downloaded using the following links:

bounding-volume hierarchies, slides:
http://cg.informatik.uni-freiburg.de/course_notes/bvh.pdf

spatial partitioning, slides:
http://cg.informatik.uni-freiburg.de/course_notes/sp.pdf

proximity queries, slides:
http://cg.informatik.uni-freiburg.de/course_notes/proximity.pdf

image-space collision detection, slides:
http://cg.informatik.uni-freiburg.de/course_notes/is.pdf

image-space collision detection, videos:
http://cg.informatik.uni-freiburg.de/movies/collision_detection_method.avi
http://cg.informatik.uni-freiburg.de/movies/collisionDetectionResultA.avi
http://cg.informatik.uni-freiburg.de/movies/collisionDetectionResultB.avi
http://cg.informatik.uni-freiburg.de/movies/collisionDetectionResultC.avi
http://cg.informatik.uni-freiburg.de/movies/collisionDetectionResultD.avi

self-collision detection, videos
http://cg.informatik.uni-freiburg.de/movies/self_collision_hand.avi
http://cg.informatik.uni-freiburg.de/movies/self_collision_torus.avi

proximity queries and spatial subdivision, videos
http://cg.informatik.uni-freiburg.de/movies/penetration_depth.avi
http://cg.informatik.uni-freiburg.de/movies/point_response.avi

fluid-deformable object interaction, video
http://cg.informatik.uni-freiburg.de/movies/fluid_deformable_interaction.avi

Collision Detection

Problem Description

Object representations in simulation environments
do not consider impenetrability.

Collision detection: Detection of interpenetrating objects.

- polygonal or non-polygonal surface
- convex, non-convex
- defined volume (closed or open surface)
- rigid or deformable objects
- pair-wise tests or multiple objects
- first contact, all contacts
- intersection, proximity, penetration depth
- static or dynamic
- discrete or continuous time

Outline

Bounding Volumes

Bounding Volume Hierarchies BVH

Generation of BVHs

Comparison

BVHs for Deformable Objects

Bounding Volumes

Simplified conservative surface representation
for fast approximative collision detection test

- **Spheres**
- **Axis-aligned bounding boxes (ABB)**
- **Object-oriented bounding boxes (OBB)**
- **Discrete orientation polytopes (k-DOPs)**

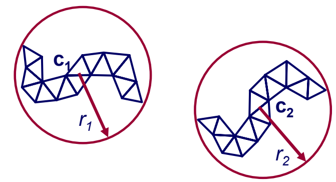
- avoid checking all object primitives.
- check bounding volumes to get the information whether objects **could** interfere. Fast rejection test.
- motivated by **spatial coherence**: Assumption that collisions between objects are rare

Requirements for Bounding Volumes

- should fit the object as tightly as possible to reduce the probability of a query object intersecting the volume but not the object
- overlap tests for bounding volumes should be efficient
- memory efficient
- efficient computation of a bounding volume, if recomputation is required

Spheres

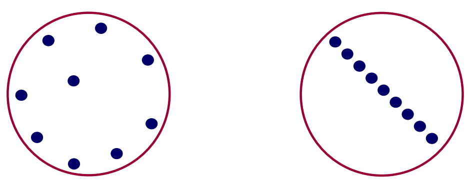
sphere is represented by center **c** and radius *r*.



two spheres do not overlap if $(\mathbf{c}_1 - \mathbf{c}_2)(\mathbf{c}_1 - \mathbf{c}_2) > (r_1 + r_2)^2$

University of Freiburg - Institute of Computer Science - Computer Graphics Laboratory

Sphere as Bounding Volume

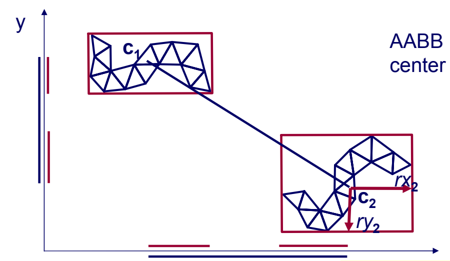


good choice

bad choice

University of Freiburg - Institute of Computer Science - Computer Graphics Laboratory

Axis-Aligned Bounding Box AABB



AABB is represented by center **c** and radii *rx*, *ry*.

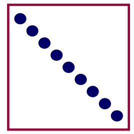
two AABBs do not overlap in 2D if $\left|(\mathbf{c}_1 - \mathbf{c}_2)\begin{pmatrix}1\\0\end{pmatrix}\right| > rx_1 + rx_2$

or $\left|(\mathbf{c}_1 - \mathbf{c}_2)\begin{pmatrix}0\\1\end{pmatrix}\right| > ry_1 + ry_2$

University of Freiburg - Institute of Computer Science - Computer Graphics Laboratory

AABB as Bounding Volume

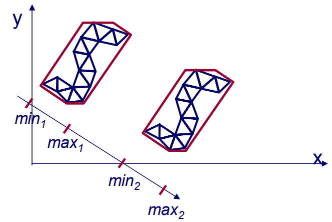
good choice



bad choice

Discrete Orientation Polytope k-DOP

A k-DOP is “a convex polytope whose facets are determined by halfspaces whose outward normals come from a small ***fixed*** set of *k* orientations.” [Klosowski]



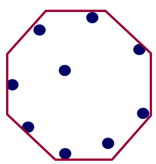
k-DOP is represented by k/2 directions and k/2 pairs of *min*, *max* values (6-, 14, 18-, 26-DOPs)

Two k-DOPs do not overlap, if their projections in at least one direction do not overlap.

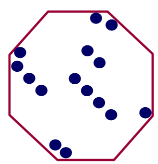
8-DOPs as Bounding Volumes

larger *k*'s are more flexible than smaller

AABB is a 4-DOP. Is a 4-DOP an AABB?

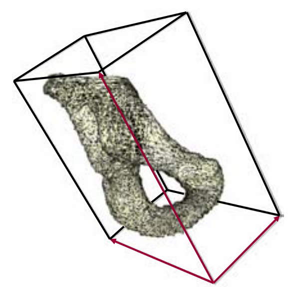


good choice



quite good choice

Oriented Bounding Box OBB



eigenvectors of the covariance matrix

An OBB can be represented by the principal axes of a set of vertices. These axes are ***not fixed***. They move according to object transformations.

vertices: $v \quad v \in \Re^3$

mean: $\mu = \frac{1}{n}\sum_{i=1}^{n} v_i$

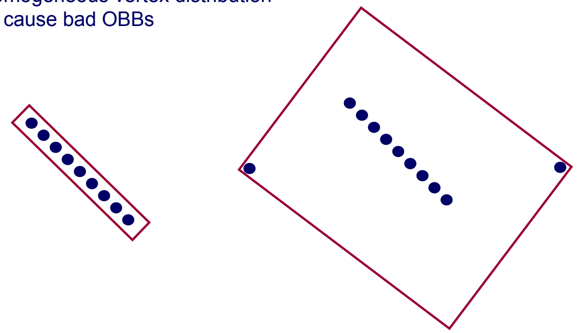
covariance matrix:

$$C_{jk} = \frac{1}{n}\sum_{i=1}^{n} \bar{v}_{ij}\bar{v}_{ik}$$

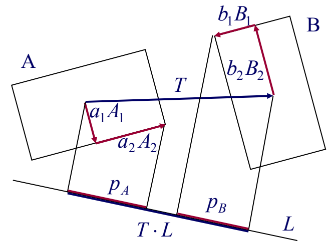
$$\bar{v}_i = v_i - \mu \qquad 1 \le j,k \le 3$$

OBB Examples

- principal axes of an object are not always a good choice for the main axes of an OBB
- inhomogeneous vertex distribution can cause bad OBBs



OBB Overlapping Test in 2D



- A_1, A_2, B_1, B_2 • axes of A, B • unit vectors
- a_1, a_2, b_1, b_2 • 'radii' of A, B
- L • unit vector

$$p_A = |a_1 A_1 L| + |a_2 A_2 L|$$

$$p_B = |b_1 B_1 L| + |b_2 B_2 L|$$

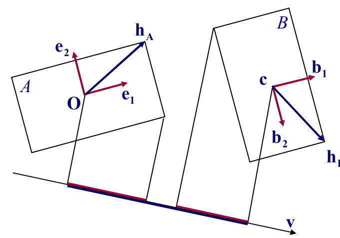
A, B do not overlap:

$\exists L : |T \cdot L| > p_A + p_B$ or $\exists L \in \{A_1, A_2, B_1, B_2\} : |T \cdot L| > p_A + p_B$

Separating Axis Test SAT

- works with polytopes: line segments, triangles, boxes
- two objects *A* and *B* are disjoint if for some vector **v** the projections of the objects onto the vector do not overlap. In this case, **v** is referred to as separating axis.
- vector **v** has to be a **face orientation** of *A* or *B* or a **cross product of two edges** of *A* and *B*.
- 3D boxes: tests with 3 + 3 + 3 · 3 axes

OBB Overlapping Test in 3D



- **B=[b$_1$ b$_2$ b$_3$]** is orientation of *B* relative to *A*'s local basis **I**
- **c** is the center of *B* relative to *A*'s local coordinate system
- **h$_A$**, **h$_B$** are the extents of *A*, *B*
- **v** is relative to *A*'s basis, **B^{T}v** is the same vector relative to *B*

- vector **v** is a separating axis iff

$$|\mathbf{v} \cdot \mathbf{c}| > |\mathbf{v}| \cdot \mathbf{h}_A + |\mathbf{B}^T \mathbf{v}| \cdot \mathbf{h}_B$$

OBB Overlapping Test in 3D

$$|\mathbf{v}\cdot\mathbf{c}| > |\mathbf{v}|\cdot\mathbf{h}_A + |\mathbf{B}^T\mathbf{v}|\cdot\mathbf{h}_B$$

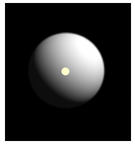
- 15 axes **v** have to be tested
  - 3 coordinate axes of *A*'s orientation **I**
  - 3 coordinate axes of *B*'s orientation **B** = [$\mathbf{b}_1$ $\mathbf{b}_2$ $\mathbf{b}_3$] = [β_{ij}]
  - 9 cross products of a coord. axis of **I** and a coord. axis of **B**
- expressions $\mathbf{B}^T\mathbf{v}$ can be simplified for all axes, e. g.

$$\mathbf{v} = \mathbf{e}_1 \times \mathbf{b}_2 = (0, -\beta_{32}, \beta_{22})^T$$
$$\mathbf{B}^T\mathbf{v} = \mathbf{B}^T(\mathbf{e}_1 \times \mathbf{b}_2) = (-\beta_{13}, 0, \beta_{11})^T$$

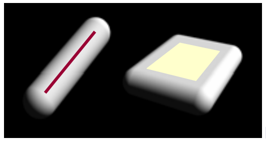
Bounding Volumes Summary

- spheres
- axis-aligned bounding boxes (AABB)
- oriented bounding boxes (OBB)
- discrete orientation polytopes (k-DOPs)

- ellipsoids
- convex Hulls
- swept-Sphere Volumes (SSVs)
  - point Swept Spheres (PSS)
  - line Swept Spheres (LSS)
  - rectangle Swept Spheres (RSS)
  - triangle Swept Spheres (TSS)



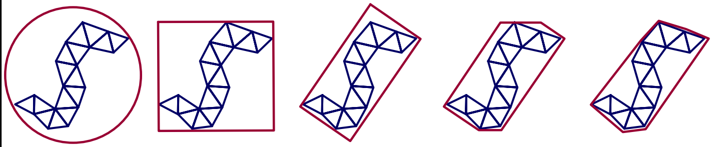
PSS



Lin, UNC

LSS RSS

Optimal Bounding Volume



sphere ABB OBB 6-DOP convex hull

tighter approximation →

← *decreasing complexity and computational expenses for overlap test*

Outline

Bounding Volumes

Bounding Volume Hierarchies BVH

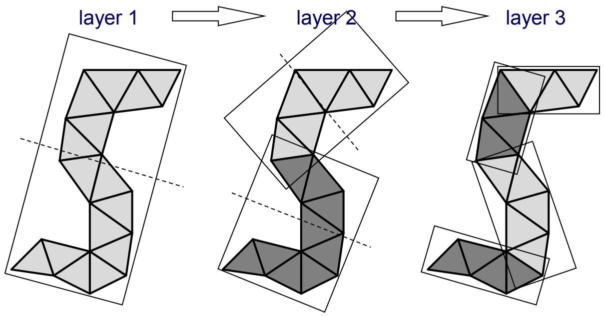
Generation of BVHs

Comparison

BVHs for Deformable Objects

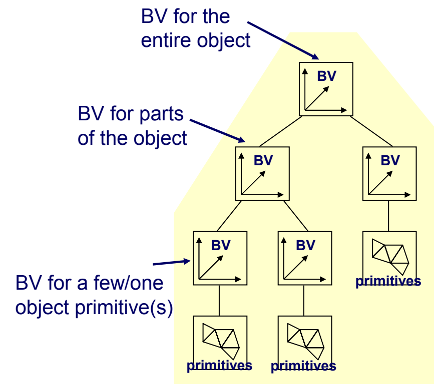
Bounding Volume Hierarchies BVHs

- subdivsion of bounding volumes to generate a hierarchy
- improved object approximation at higher levels

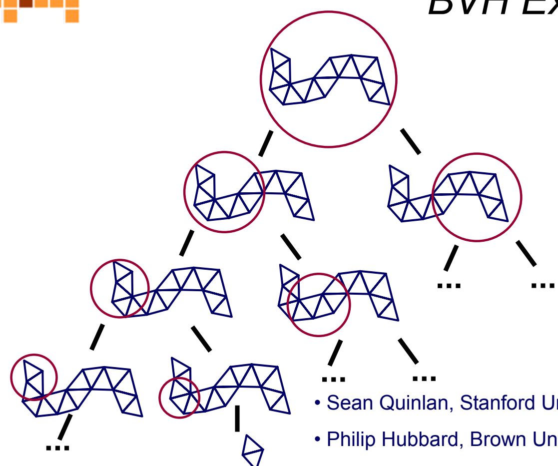


Hierarchy of Bounding Volumes

- bounding volume tree (BV tree)
- nodes contain bounding volume information
- leaves additionally contain information on object primitives

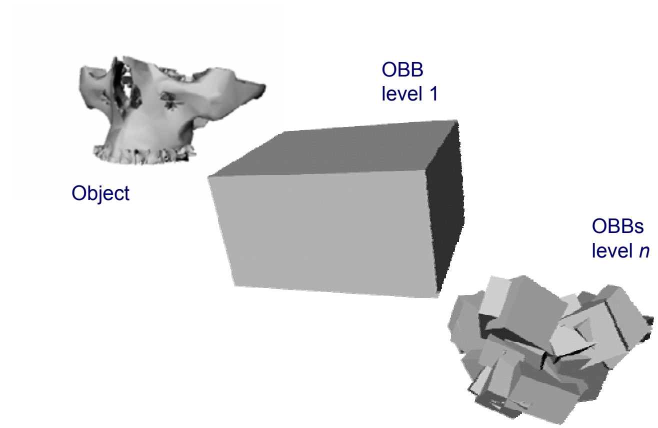


BVH Example



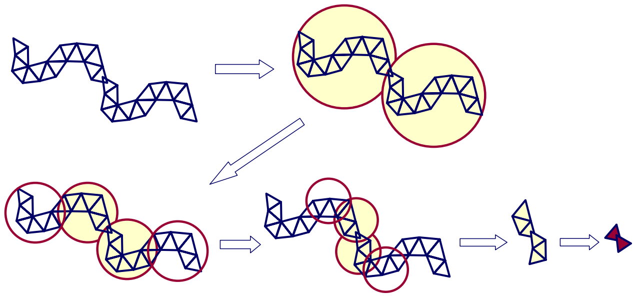
- Sean Quinlan, Stanford Univ
- Philip Hubbard, Brown Univ

OBB Tree



Overlapping Test for BV Tree

- BV-trees speed-up the collision detection test
- if bounding volumes in a hierarchy level overlap, their children are checked for overlapping. If leaves are reached, primitives are checked against each other.



Overlapping Test for BV Tree

Pseudo code

1. interference check for two parent nodes (root)
2. if no interference then “no collision” else
3. all children of one parent node are checked against children of the other parent node
4. if no interference then “no collision” else
5. if at leave nodes then “collision” else go to 3

step 3 checks BVs or object primitives for intersection

Box-Triangle and Triangle-Triangle Test

Box-Triangle Test

a) separating axes test requires 13 axes to be tested (4 face normals, 3 x 3 cross products of edges)

Triangle-Triangle Test

a) separating axes test requires max. 11 axes to be tested (2 face normals, 3 x 3 cross products of edges)

b) testing each edge of one triangle against the other triangle for intersection -> 6 edge-triangle tests

(edge-triangle intersections occur in pairs -> 5 tests are sufficient)

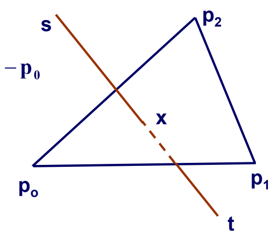
Edge-Triangle Test

$$\mathbf{x} = \mathbf{p_0} + \mu_1(\mathbf{p_1} - \mathbf{p_0}) + \mu_2(\mathbf{p_2} - \mathbf{p_0}) \quad \mu_1, \mu_2 \geq 0 \quad \mu_1 + \mu_2 \leq 1$$

$$\mathbf{x} = \mathbf{s} + \lambda(\mathbf{t} - \mathbf{s}) \quad 0 \leq \lambda \leq 1$$

$$\mathbf{r} = \mathbf{t} - \mathbf{s} \quad \mathbf{d}_1 = \mathbf{p}_1 - \mathbf{p}_0 \quad \mathbf{d}_2 = \mathbf{p}_2 - \mathbf{p}_0 \quad \mathbf{b} = \mathbf{s} - \mathbf{p}_0$$

$$\mathbf{b} = \mu_1\mathbf{d}_1 + \mu_2\mathbf{d}_2 - \lambda\mathbf{r}$$



$$\begin{pmatrix}\lambda \\ \mu_1 \\ \mu_2\end{pmatrix} = \frac{1}{-\mathbf{r}\cdot(\mathbf{d}_1 \times \mathbf{d}_2)}\begin{pmatrix}\mathbf{b}\cdot(\mathbf{d}_1 \times \mathbf{d}_2) \\ \mathbf{d}_2 \cdot (\mathbf{b} \times \mathbf{r}) \\ -\mathbf{d}_1 \cdot (\mathbf{b} \times \mathbf{r})\end{pmatrix}$$

edge intersects iff

$$-\mathbf{r}\cdot(\mathbf{d}_1 \times \mathbf{d}_2) \neq 0 \quad 0 \leq \lambda \leq 1 \quad \mu_1 + \mu_2 \leq 1 \quad \mu_1, \mu_2 \geq 0$$

Characteristics of BVH

- improved object approximation at higher levels
- fast rejection query
- fast localization of object regions with potential collisions

- additional storage requirements
- generation of BVHs can be expensive
  - BVHs are generally used for rigid models where they can be pre-computed

Computational Costs of BV Trees

Cost function (M. Lin, UNC):

$$F = N_u \times C_u + N_{bv} \times C_{bv} + N_p \times C_p$$

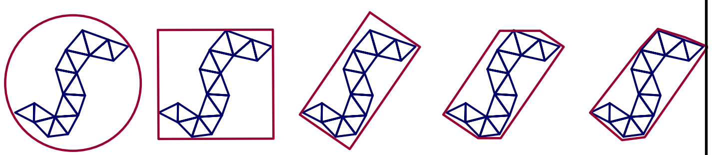
tree generation/update — BV intersection test — primitive intersection test

F:	total cost for interference detection
N_u:	number of bounding volumes updated
C_u:	cost of updating a bounding volume
N_{bv}:	number of bounding volume pair overlap tests
C_{bv}:	cost of overlap test between two bounding volumes
N_p:	number of primitive pairs tested for interference
C_p:	cost of testing two primitives for interference

Optimization

$$F = N_u \times C_u + N_{bv} \times C_{bv} + N_p \times C_p$$

- infrequent BV updates to minimize N_u
- tight-fitting bounding volumes to minimize N_{bv}
- simple intersection test for bounding volumes to minimize C_{bv}

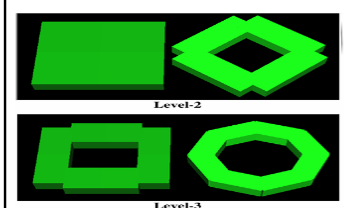


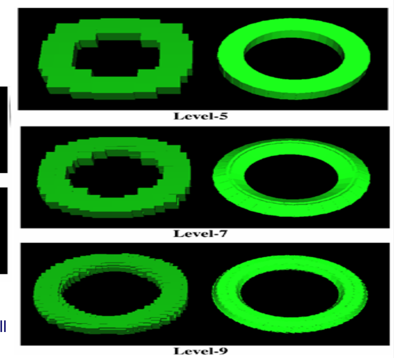
Better approximation →

← *Decreasing computational expenses for overlap test*

AABB vs. OBB Tree

approximation of a torus





Lin, UNC Chapel Hill

Object Transformations

some object transformations can be simply applied
to all elements of the bounding-volume tree:

Spheres

- translation, rotation

Axis-Aligned Bounding Boxes

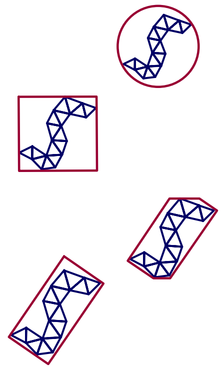
- translation, no rotation

Discrete Orientation Polytopes

- translation, no rotation
  (principal orientations are fixed for all objects)

Object-Oriented Bounding Boxes

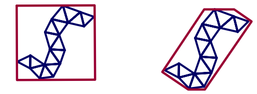
- translation, rotation
  (box orientations are not fixed)



Rotations

Axis-Aligned Bounding Boxes

Discrete Orientation Polytopes



- rotation of the bounding volume is not
  possible due to the respective box overlap test.
  The intersection tests require fixed surface normals.

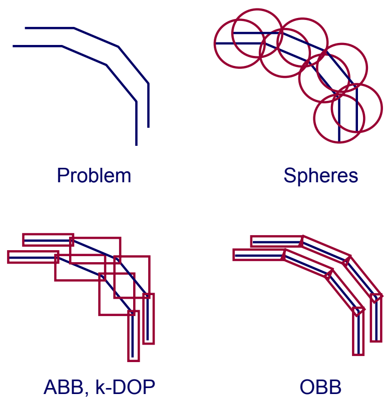
1. recomputation of the BV hierarchy
2. preservation of the tree structure, update of all nodes

a) additional storage of the convex hull which is rotated with the object
   - check if extremal vertices are still extremal after rotation
   - compare with adjacent vertices of the convex hull
   - “climb the hill” to the extremal vertex

b) computation of an approximate box by rotating the box and
   checking the rotated box for extremal values

Close Proximity

- quality of ***higher***-level
  BV approximation
  influences collision
  detection performance
  in case of close proximity
- quality of higher-level
  BV approximations is
  not very critical
- in case of overlapping BV
  expensive primitive tests
  have to be performed



Outline

Bounding Volumes

Bounding Volume Hierarchies BVH

Generation of BVHs

Comparison

BVHs for Deformable Objects

Construction of a BV Tree

Bottom-Up

- start with object-representing primitives
- fit a bounding volume to each primitive
- group primitives or bounding volumes recursively
- fit bounding volumes to these groups
- stop in case of a single bounding volume at a hierarchy level

Top-Down

- start with object
- fit a bounding volume to the object
- split object or bounding volume recursively
- fit bounding volumes
- stop, if all bounding volumes in a level contain less than *n* primitives

Construction of a BV Tree

Parameters

- bounding volume
- top-down vs. bottom-up
- what to subdivide / group: object primitives or bounding volumes
- how to subdivide / group object primitives or bounding volumes
- how many primitives in each leaf of the BV tree
- re-sampling of the object ?

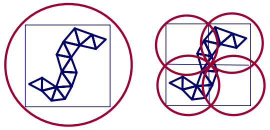
Goals

- balanced tree
- tight-fitting bounding volumes
- minimal redundancy
  (primitives in more than one BV per level)

Construction of a BV Tree Spheres

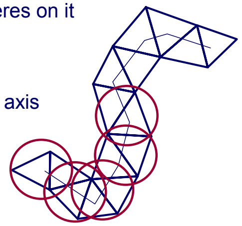
Hubbard, C. O'Sullivan:

- approximate triangles with spheres and build the tree bottom-up by grouping spheres
- cover vertices with spheres and group them
- resample vertices prior to building the tree
  (homogeneous vertex distribution reduces redundancy)
- build the tree top-down by using an octree
- compute the medial axis and place spheres on it



octree based

medial axis based



Outline

Bounding Volumes

Bounding Volume Hierarchies BVH

Generation of BVHs

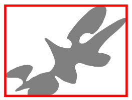
Comparison

BVHs for Deformable Objects

Collision Detection Libraries

SOLID

Axis-aligned bounding box



van den Bergen
Eindhoven University
1997

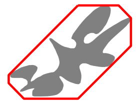
RAPID

Object-oriented bounding box

Gottschalk et al.
University of North Carolina
1995

QuickCD

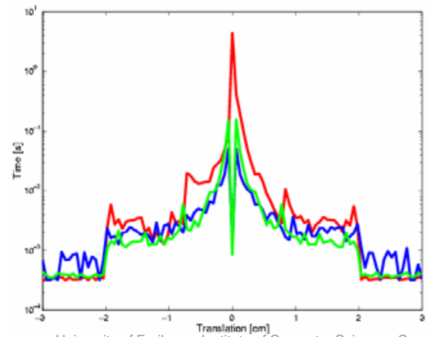
k discrete orientation polytope

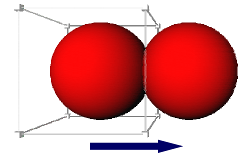


Klosowski et al.
University of New York
1998

Comparison of CD Libraries

- time to compute a collision for two spheres with radius 1 cm
- translation represents the distance of both centers
- QuickCD [Klosowski], RAPID [Gottschalk], SOLID [Bergen]





10,000 triangles per sphere
8-DOP
OBB
ABB

Outline

Bounding Volumes

Bounding Volume Hierarchies BVH

Generation of BVHs

Comparison

BVHs for Deformable Objects

BVHs for Deformable Collision Detection

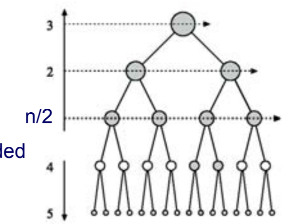
- in case of deformable objects, BVH has to be updated frequently
- hierarchy generation significantly influences performance
- AABBs are commonly used
- AABBs can be updated efficiently compared to OBB, k-DOP, spheres
- however, AABBs do not provide an optimal model approximation

Hybrid Hierarchy Update

- proposed by Larsson / Akenine-Moeller, Eurographics 2001
- AABB hierarchy
- initial hierarchy generation as pre-processing

- lazy hierarchy update during run-time
    - bottom-up update starting at depth n/2
    - very efficient AABB update based on AABBs of children

- update of nodes in depth n/2+1 to n as needed
- this update is only performed if necessary

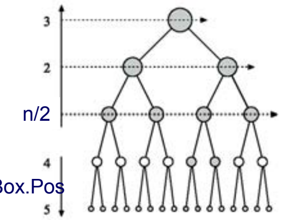


Implementation of Hierarchy Update

- after pre-processing each nodes knows which vertices influence its bounding box

- object is traversed once to update nodes (box information) in layer n/2

- bottom-up merging of AABBs
    - Merge (b1, b2)
    Box.Pos = Min(b1.Pos, b2.Pos)
    Box.Size = Max(b1.Pos+b1.Size, b2.Pos+b2.Size)-Box.Pos



Hierarchical Bounding Volumes - Summary

- bounding volume tree (BV tree) based on spheres or boxes
- nodes contain bounding volume information
- leaves additionally contain information on object primitives

- isolating interesting regions by checking bounding volumes in a top-down strategy

- construction of a balanced, tight-fitting tree with minimal redundancy

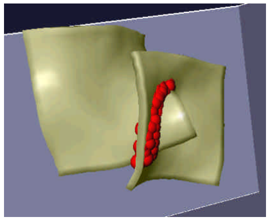
- transformation of BV trees dependent on the basic bounding volume

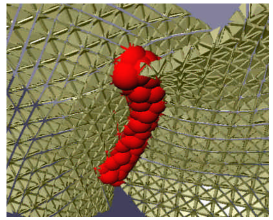
- optimal bounding box hierarchy dependent on application (e. g. close proximity problem)

References

- **S. Quinlan**, “Efficient Distance Computation Between Non-Convex Objects,” *Proc. Int. Conf. on Robotics and Automation*, pp. 3324-3329, 1994.
- **P. M. Hubbard**, “Approximating Polyhedra With Spheres for Time-Critical Collision Detection,” *ACM ToG*, 15 / 3, pp. 179-210, 1996.
- **S. Gottschalk**, M. C. Lin, D. Manocha, “OBBTree: A Hierarchical Structure for Rapid Interference Detection,” *Proc. SIGGRAPH’96, ACM Computer Graphics*, New York, NY, USA, pp. 171-180, 1996.
- **G. van den Bergen**, “Efficient Collision Detection of Complex Deformable Models using AABB Trees,” *Journal of Graphics Tools*, 2 / 4, pp. 1-13, 1997.
- **J. T. Klosowski** et al., “Efficient Collision Detection Using Bounding Volume Hierarchies of k-DOPs,” *IEEE Trans on Vis and Computer Graphics*, 4 / 1, pp. 21-36, 1998.
- **Larsson, Akenine-Moeller**, “Collision Detection for Continuously Deforming Objects,” *Proc. Eurographics,* 2001.
- **G. van den Bergen**, “*Collision Detection in Interactive 3D Environments*,” Elsevier, Amsterdam, ISBN: 1-55860-801-X, 2004.

Collision Detection - Spatial Partitioning





Acknowledgements

- Parts of this slide set are courtesy of Bruno Heidelberger, ETH Zurich.

University of Freiburg - Institute of Computer Science - Computer Graphics Laboratory

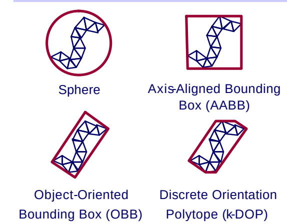
Outline

- introduction to spatial data structures
- binary space partitioning trees
- voxel grids
- spatial subdivision with graphics hardware

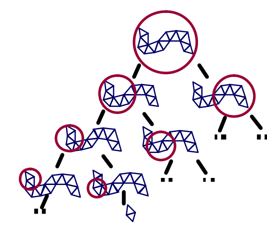
University of Freiburg - Institute of Computer Science - Computer Graphics Laboratory

Bounding Volume Hierarchies

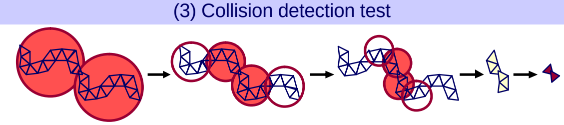
(1) Bounding volumes



(2) Bounding volume tree



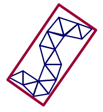
(3) Collision detection test



University of Freiburg - Institute of Computer Science - Computer Graphics Laboratory

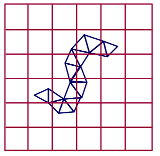
BVHs vs. Spatial Partitioning

Bounding Volume Hierarchy



Model partitioning

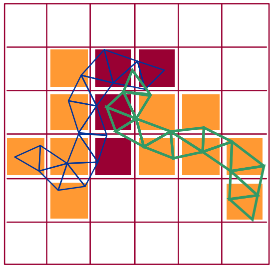
Spatial Partitioning



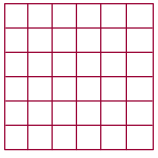
Space partitioning

Spatial Partitioning - Idea

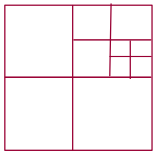
- space is divided up into cells
- object primitives are placed into cells
- object primitives within the same cell are checked for collision
- pairs of primitives that do not share the same cell are not tested (trivial reject)



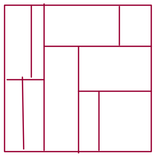
Spatial Data Structures



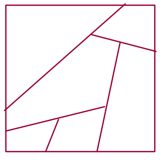
voxel grid



octree



k-d tree



BSP-tree

- cells maintain references to primitives intersecting the cell
- information is updated for each object transformation
- octree, k-d tree, and BSP-tree are object-dependent
- voxel grid is object-independent

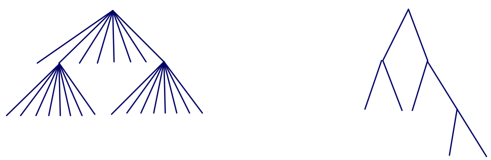
Voxel Grid

- space partitioning into (uniform) rectangular, axis-aligned cells
- primitives per cell are found by
  - scan conversion of primitives to the grid or
  - scan conversion of AABBs of the primitives
- fast cell access
- optimal cell size?
  - large cells increase the number of primitives per cell
  - small cells cause spreading of primitives to a large number of cells
- less efficient in case of non-uniform primitive distribution

Octree and k-d Tree

- hierarchical structures
- space partitioning into rectangular, axis-aligned cells
- root node corresponds to AABB of an object
- internal nodes represent subdivisions of the AABB
- leaves represent cells which maintain primitive lists

octree k-dimensional binary tree



Octree and k-d Tree

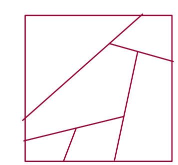
- uniform or non-uniform subdivision
- adaptive to local distribution of primitives
  - large cells in case of low density of primitives
  - small cells in case of high density
- dynamic update
  - cells with many primitives can be subdivided
  - cells with less primitives can be merged

Outline

- introduction to spatial data structures
- binary space partitioning trees
- voxel grids

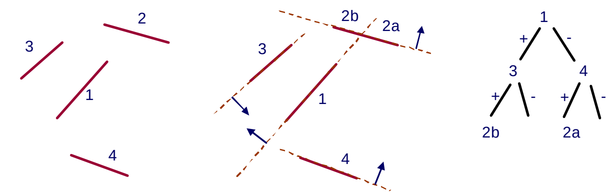
BSP Tree

- binary space partitioning tree
- hierarchical structure
- space is subdivided by means of arbitrarily oriented planes
- generalized k-d tree
- space partitioning into convex cells
- discrete-orientation BSP trees DOBSP (finite set of plane orientations)



BSP Tree for Rendering

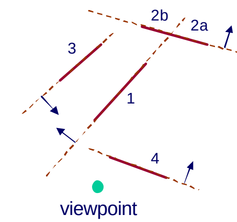
- [Henry Fuchs et al. 1980] proposed
  a visible surface algorithm using a pre-computed BSP

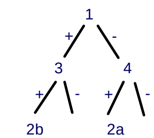


original scene | scene partitioning | BSP tree

BSP Tree for Rendering

- for a given viewpoint
  - render far branch
  - render root (node) polygon
  - render near branch
- recursively applied to sub-trees
- back to front rendering
- example: viewpoint is in 1-
- rendering of 1+, 1, 1-
- rule recursively applied to 1+ and 1-
- viewpoint is in 3+ -> rendering of 3, 2b
- viewpoint is in 4- -> rendering of 2a, 4

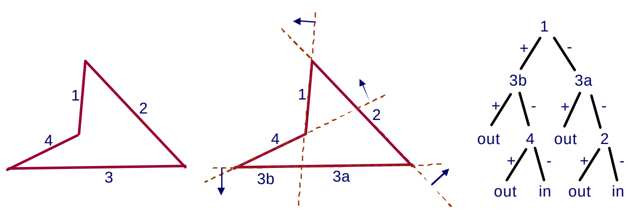




BSP tree

BSP Tree for Collision Detection

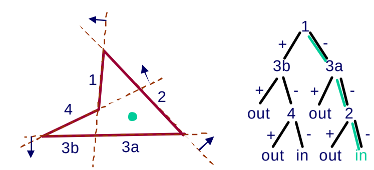
- BSP trees can be used for
  inside / outside classification of
  closed polygons



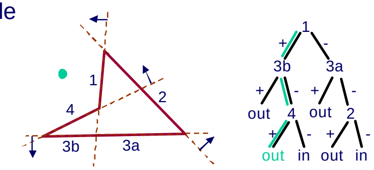
original scene | scene partitioning | solid-leaf BSP tree

Collision Query

- query point is inside



- query point is outside



BSP Tree Construction

- keep the number of nodes small
- keep the number of levels small
- introduce arbitrary support planes
  (especially in case of convex objects,
  where all polygon faces are in the same half-space
  with respect to a given face)

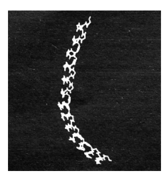
Outline

- introduction to spatial data structures
- binary space partitioning trees
- voxel grids

University of Freiburg - Institute of Computer Science - Computer Graphics Laboratory

Related Approaches

- [Levinthal 1966]
  - 3D grid ("cubing")
  - analysis of molecular structures
  - neighborhood search
    to compute atom interaction
- [Rabin 1976]
  - 3D grid + hashing
  - finding closest pairs
- [Turk 1989, 1990]
  - rigid collision detection
  - 3D grid + hashing

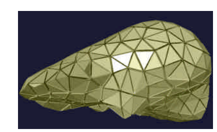


Cyrus Levinthal, MIT

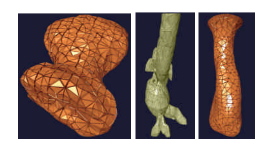
University of Freiburg - Institute of Computer Science - Computer Graphics Laboratory

Deformable Collision Detection

- [Teschner, Heidelberger et al. 2003]
  - collisions and self-collisions for
    deformable tetrahedral meshes
  - uniform 3D grid
  - non-uniform distribution
    of object primitives
    → hashing
  - no explicit 3D data structure
  - analysis of optimal cell size



Epidaure, INRIA

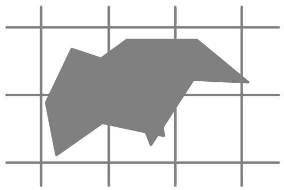


NCCR Co-Me

University of Freiburg - Institute of Computer Science - Computer Graphics Laboratory

Algorithm - Setup

implicit uniform grid:

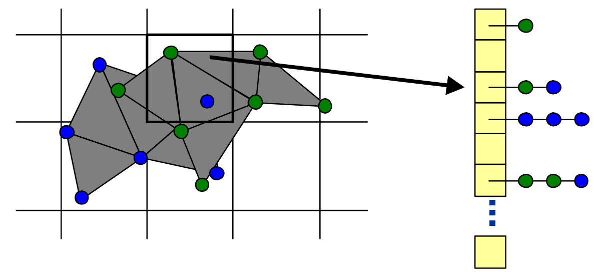


hash function:
H(cell) → hash table index

hash table:

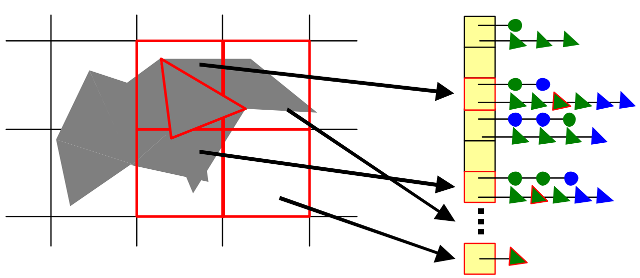
Algorithm – Stage 1

- all vertices are hashed according to their cell:



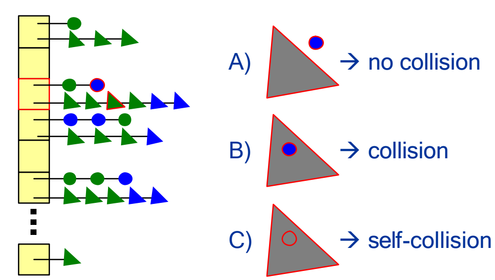
Algorithm – Stage 2

- all tetrahedrons are hashed according to the cells touched by their bounding box



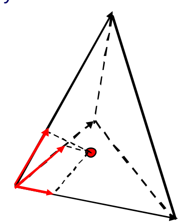
Algorithm – Stage 3

- vertices and tetrahedrons in the same hash table entry are tested for intersection:

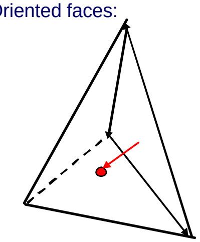


Vertex-in-Tetrahedron Test

(a) Barycentric coordinates:



(b) Oriented faces:



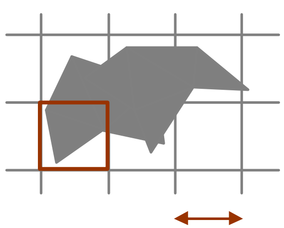
→ Barycentric coordinates faster

Algorithm – Summary

- stages:
  - hash all vertices
  - hash all tetrahedrons
  - intersection test within each hash table entry
- parameters:
  - grid cell size
  - grid cell shape
  - hash table size
  - hash function

Algorithm - Parameters

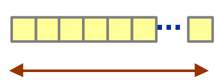
implicit uniform grid:



hash function:

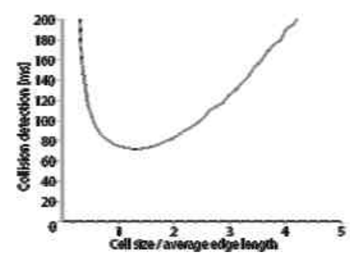
H(cell) → hash table index

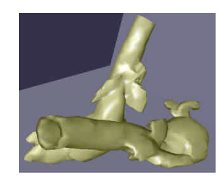
hash table:



Grid Cell Size

- [Bentley et al. 1977] suggest a cell size equal to the size of the bounding box of an object primitive
- [Teschner, Heidelberger et al. 2003]

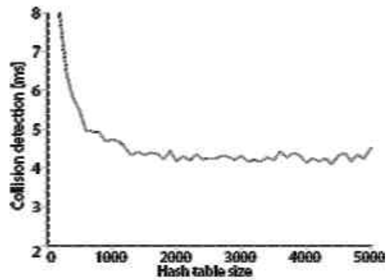


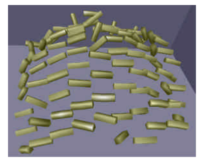


test scenario

Hash Table Size

- larger hash table reduces hash collisions





test scenario

Hash Function

H(i, j, k) := (i · p_1 ***xor*** j · p_2 ***xor*** k · p_3) ***mod*** n

i, j, k : cell coordinates

p_1, p_2, p_3 : large primes

n : hash table size

Performance

- [Teschner, Heidelberger et al. 2003]
  collision and self-collision detection

objects	tetras	vertices	max time [ms]
100	1000	1200	6
8	4000	1936	15
20	10000	4840	34
2	20514	5898	72
100	50000	24200	174

Pentium 4, 1.8GHz

test scenarios

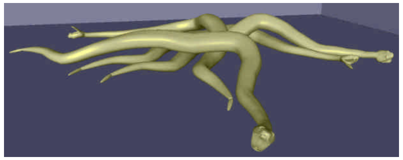
Uniform Voxel Grids

- collision and self-collision detection of tetrahedral meshes
- no explicit spatial partitioning
  (AABB and cells are not explicitly represented)
- hash map
- performance dependent on number of object primitives
- performance independent of number of objects

- algorithm can work with various object primitives

Uniform Voxel Grids

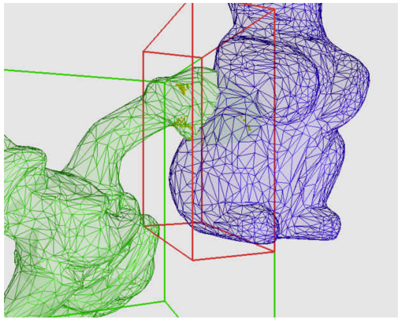
- simple and efficient technique
- especially interesting for deformable, n-body, and self-collision detection
- in case of non-uniform or sparse spatial distribution of object primitives, hashing is a good choice
- parameters have to be investigated



References

- C. Levinthal, "Molecular model-building by computer," Scientific American, pp. 42-52, June 1966.
- J. L. Bentley, D. F. Stanat, E. H. Williams, "The complexity of fixed-radius near neighbor searching," *Inf. Process. Letters*, vol. 6, 209-212, 1977.
- G. Turk, "Interactive collision detection for molecular graphics," TR90-014, University of North Carolina at Chapel Hill, 1990.
- S. Bandi, D. Thalmann, "An adaptive spatial subdivision of the object space for fast collision detection of animating rigid bodies," *Proc. of Eurographics*, pp. 259-270, 1995.
- A. Gregory, M. Lin, S. Gottschalk, R. Taylor, "*H-COLLIDE: A framework for fast and accurate collision detection for haptic interaction*," TR98-032, University of North Carolina at Chapel Hill, 1998.
- S. Melax, "Dynamic plane shifting BSP traversal," *Proc. Graphics Interface*, pp. 213-220, 2000.
- M. Teschner, B. Heidelberger, M. Mueller, D. Pomeranets, M. Gross, "Optimized Spatial Hashing for Collision Detection of Deformable Objects," *Proc. Vision, Modeling, Visualization VMV'03*, pp. 47-54, Nov 2003.
- G. van den Bergen, "*Collision Detection in Interactive 3D Environments*," Elsevier, Amsterdam, ISBN: 1-55860-801-X, 2004.

Image-Space Collision Detection



Acknowledgements

- Parts of this slide set are courtesy of Bruno Heidelberger, ETH Zurich.

University of Freiburg - Institute of Computer Science - Computer Graphics Laboratory

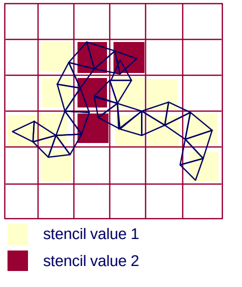
Outline

- motivation
- algorithms
- performance
- application
- discussion

University of Freiburg - Institute of Computer Science - Computer Graphics Laboratory

Graphics Hardware for 2D Collision Detection

frame buffer is a uniform grid

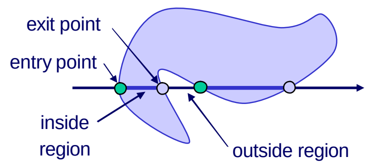


- Kenneth Hoff, UNC
- stencil-buffer for collision detection

- clear stencil buffer
- increment stencil buffer for each rendered object
- intersection for stencil buffer value larger 1

University of Freiburg - Institute of Computer Science - Computer Graphics Laboratory

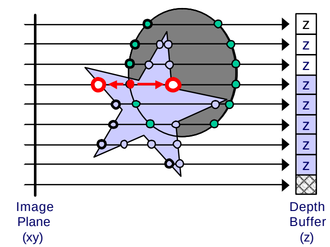
Closed Objects



- number of entry points equals the number of exit points
- in case of convex objects, one entry point and one exit point
- inside and outside are separated by entry or exit point
- entry point is at a front face
- exit point is at a back face
- front and back faces alternate

Collision Detection with Graphics Hardware

- exploit rasterization of object primitives as intersection test
- benefit from graphics hardware acceleration



Collision Detection with Graphics Hardware

Idea

- computation of entry and exit points can be accelerated with graphics hardware
- computation corresponds to rasterization of surface primitives
- all object representations that can be rendered are handled
- parallel processing on CPU and GPU

Challenges

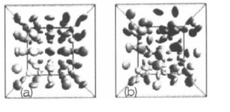
- restricted data structures and functionality

Drawbacks

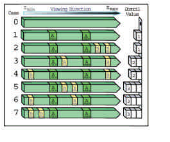
- approximate computation of entry and exit points

Early approaches

[Shinya, Forgue 1991]
image-space collision detection for *convex objects*



[Myszkowski, Okunev, Kunii 1995]
collision detection for *concave objects with limited depth complexity*



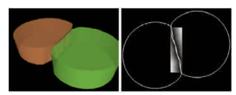
[Baciu, Wong 1997]
hardware-assisted collision detection for *convex objects*

More approaches

[Lombardo, Cani, Neyret 1999]
intersection of *tool with deformable tissue* by rendering the interior of the tool

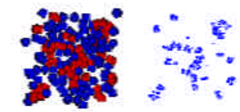
[Vassilev, Spanlang, Chrysanthou 2001]
image-space collision detection applied to cloth simulation and *convex avatars*

[Hoff, Zaferakis, Lin, Manocha 2001]
proximity tests and penetration depth computation, *2D*

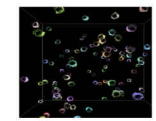


Recent approaches

[Knott, Pai 2003]
intersection of edges with surfaces



[Govindaraju, Redon, Lin, Manocha 2003]
object and sub-object pruning based on occlusion queries



[Heidelberger, Teschner 2004]
explicit intersection volume and self-collision detection based on LDIs

Image-Space Collision Detection [Knott, Pai 2003]

- render all query objects (e. g. edges) to depth buffer
- count the number *f* of front faces that occlude the query object
- count the number *b* of back faces that occlude the query object
- iff *f* - *b* == 0 then there is no collision

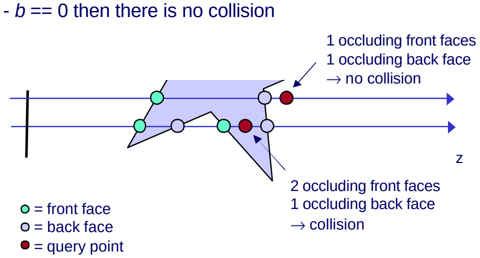


Image-Space Collision Detection

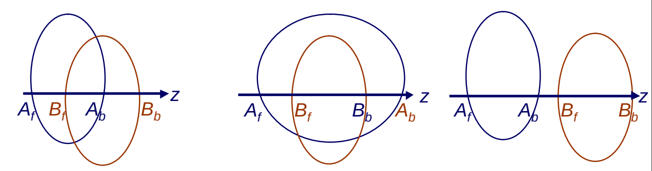
- clear depth buffer, clear stencil buffer
- render query objects to depth buffer
- disable depth update
- render front faces with stencil increment
  - if front face is closer than query object, then stencil buffer is incremented
  - depth buffer is not updated
  - result: stencil buffer represents number of occluding front faces
- render back faces with stencil decrement
  - if back face is closer than query object, then stencil buffer is decremented
  - depth buffer is not updated
  - result: stencil buffer represents difference of occluding front and back faces
- stencil buffer not equal to zero → collision

Image-Space Collision Detection

- works for objects with closed surface
- works for n-body environments
- works for query objects that do not overlap in image space
- numerical problems if query object is part of an object
  - offset in *z*-direction required
- [Video]

Image-Space Collision Detection [Baciu 2000]

- RECODE – REndered COllision DEtection
- works with pairs of closed convex objects A and B
- one or two rendering passes for A and B
- algorithm estimates overlapping *z* intervals per pixel



collision | collision | no collision

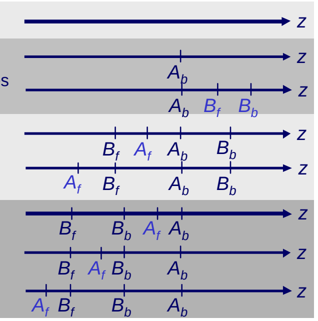
First Rendering Pass

- clear depth buffer
- clear stencil buffer
- enable depth update
- render back faces of A with stencil increment
  - if nothing has been rendered → stencil=0
  - if something has been rendered → stencil=1
  - depth buffer contains depth of back faces of A
- disable depth update
- render B with stencil increment
  - if stencil==1 and B occludes back face of A → stencil+=1
  - depth buffer is not updated
  - stencil-1 = number of faces of B that occlude A

First Rendering Pass

- first pass collision query

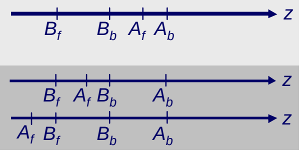
- stencil 0 → no collision
- stencil 1 → no collision
  - no fragment of B occludes back face of A (2 cases)
- stencil 2 → collision
  - front face of B occludes back face of A (2 cases)
- stencil 3 → second pass
  - front and back face of B occlude back face of A (3 cases)



Second Rendering Pass

- render back faces of object B, count occluding faces of A
  - corresponds to first pass with A and B permuted
  - only 3 cases based on the result of the first rendering pass

- stencil 1 → no collision
  - no fragment of A occludes back face of B (1 case)
- stencil 2 → collision
  - front face of A occludes back face of B (2 cases)

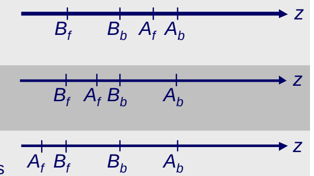


- done

Second Rendering Pass [Myszkowski 1995]

- render front faces of object A, count occluding faces of B
  - corresponds to first pass, front faces are rendered instead of back faces
  - only 3 cases based on the result of the first rendering pass

- stencil 3 → no collision
  - front and back face of B occlude front face of A
- stencil 2 → collision
  - front face of B occludes front face of A
- stencil 1 → collision
  - no fragment of B occludes front face of A



- done

Image-Space Collision Detection for Concave Objects [Myszkowski 1995]

- collision detection for pairs of concave objects
  A and B with limited depth complexity (number of entry/exit points)
- faces have to be sorted with respect
  to the direction of the orthogonal projection (e. g. BSP tree)

- objects are rendered in front-to-back or back-to-front order
- alpha blending is employed:
  $$color_{framebuffer} = color_{object} + \alpha \cdot color_{framebuffer}$$
- color of A is zero, color of B is 2^{k-1},
  k is the number of bits in the frame buffer,
  $\alpha = 0.5$

Image-Space Collision Detection for Concave Objects

- example: $k = 8$
- color A = 0, color B = 2^7
- sequence of faces B_1 A_1 A_2 B_2 B_3 B_4 rendered back to front:
  - $c_{fb} = 00000000_2$
  - render B_4: $c_{fb} = 2^7 + \alpha \cdot c_{fb} = 10000000_2 + 0.5 \cdot 00000000_2 = 10000000_2$
  - render B_3: $c_{fb} = 10000000_2 + 0.5 \cdot 10000000_2 = 11000000_2$
  - render B_2: $c_{fb} = 10000000_2 + 0.5 \cdot 11000000_2 = 11100000_2$
  - render A_2: $c_{fb} = 00000000_2 + 0.5 \cdot 11100000_2 = 01110000_2$
  - render A_1: $c_{fb} = 00000000_2 + 0.5 \cdot 01110000_2 = 00111000_2$
  - render B_1: $c_{fb} = 10000000_2 + 0.5 \cdot 00111000_2 = 10011100_2$
- resulting bit sequence represents order of faces of A (0) and B (1)
- odd number of adjacent zeros or ones indicates collision

Image-Space Collision Detection for Concave Objects

- example:

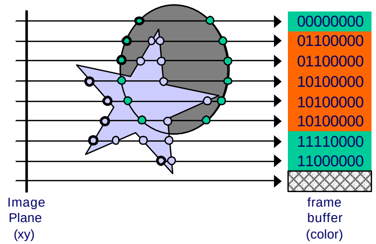
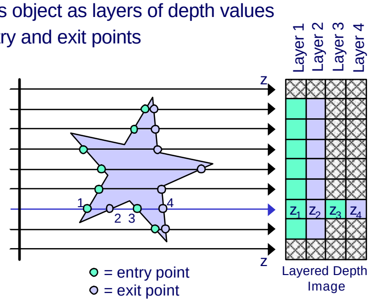


Image-Space Collision Detection [Heidelberger 2003]

- works with pairs of closed arbitrarily-shaped objects
- three implementations
  - *n+1* hardware-accelerated rendering passes where n is the depth complexity of an object
  - *n* hardware-accelerated rendering passes
  - 1 software rendering pass
- three collision queries
  - intersection volume (based on intersecting *z* intervals)
  - vertex-in-volume test
  - self-collision test
- basic idea and implementation for convex objects has been proposed by Shinya / Forgue in 1991

Layered Depth Image

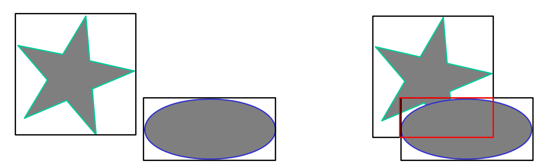
- compact, volumetric object representation [Shade et al. 1998]
- represents object as layers of depth values
- stores entry and exit points



Algorithm Overview

Algorithm consists of 3 stages:

Stage 1: Check for bounding box intersection

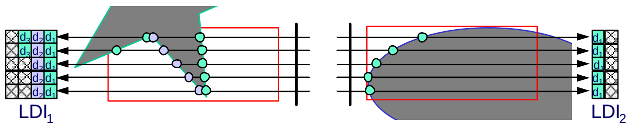


a) Very fast detection of trivial "no collision" cases

b) Overlapping area defines volume of interest (VoI) for step 2 & 3

Algorithm Overview

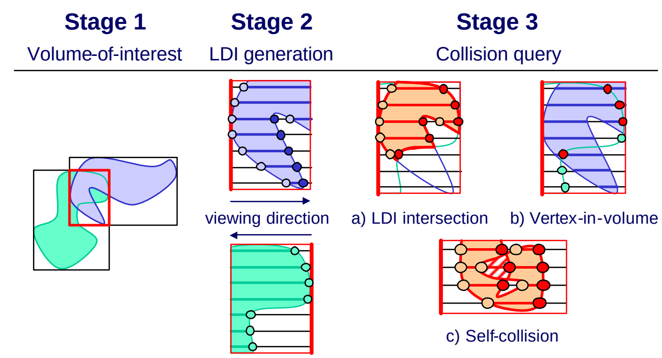
Stage 2: Generate the layered depth images (LDI)



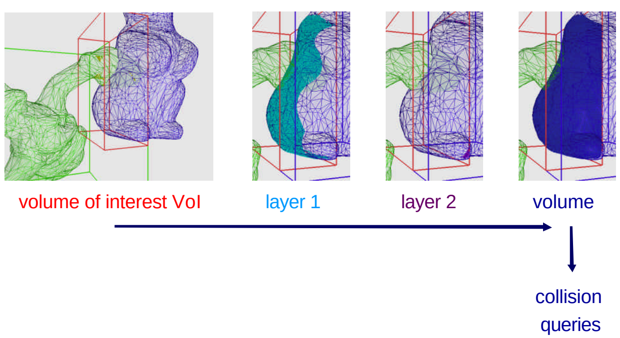
Step 3: Perform the collision tests

a) test object primitives of one object against LDI of the other
b) combine both LDI to get overlapping volume
c) self-intersection test

Algorithm Overview



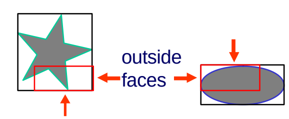
Algorithm Overview



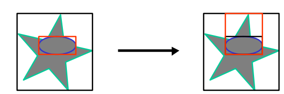
Volume of Interest

VoI = BoundingBox(Object 1) ∩ BoundingBox(Object 2)

1. evaluation of trivial rejection test: VoI == Ø → no collision!
2. choice of *opposite* render directions for LDI generation



possible enlargement of VoI to guarantee valid directions

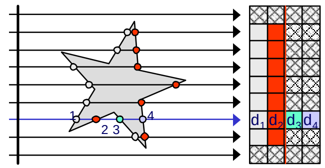


outside faces are outside the object
-> guarantees that first intersection point is an entry point

LDI Generation on the GPU
Depth Peeling

- object is rendered once for each layer in the LDI
- two separate depth tests per fragment are necessary:
  - fragment must be farther than the one in the previous layer (**d_2**)
  - fragment must be the nearest of all remaining fragments (**d_3** & **d_4**)

example: *pass #3*



→ second depth test is realized using shadow mapping
extended depth-peeling approach [Everitt 2001]

Shadow Mapping

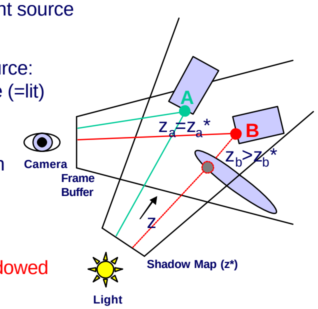
Idea:

- for each fragment to be rendered: check if it is visible from the light source

Algorithm:

- render scene from the light source: store all distances to the visible (=lit) fragments in a “shadow map”
- render scene from the camera: compare the distance *z* of each fragment to the light with the value *z** in the shadow map:

  $z = z^*$ → fragment is lit
  $z > z^*$ → fragment is shadowed



Shadow Mapping
as Depth Test

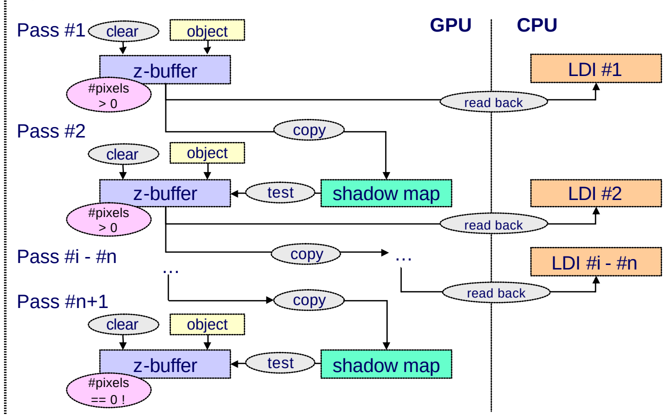
Differences to regular depth test:

- shadow mapping depth test is not tied to camera position
- shadow map (depth buffer) is *not writeable during depth test*
- shadow mapping *does not discard fragments*

Depth test setup for LDI generation:

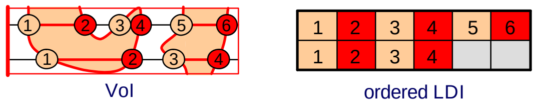
- fragment must be farther away than fragment in previous depth layer → shadow map test
- fragment must be the nearest of all remaining fragments → regular depth test

Multipass LDI Generation



Result of LDI Generation

- multipass LDI generation results in an ordered LDI representation of the VoI

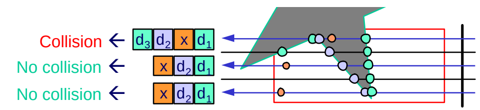


- requires one rendering pass per depth layer
- requires shadow mapping functionality

Collision Detection Test

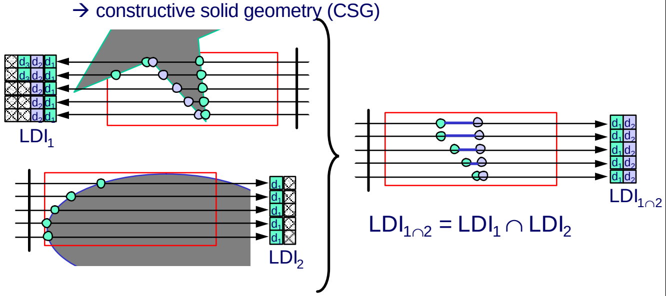
- test object primitives of one object against LDI of the other object (and vice versa)
- vertex-in-volume test

example:



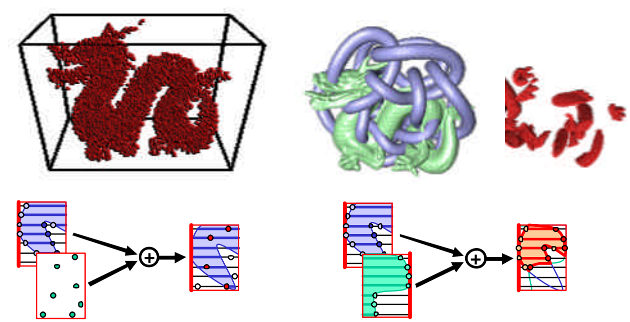
LDI Combination

- intersect both LDI to get the overlapping volume
- provides an explicit intersection volume
- other boolean operations (union, difference) are also possible → constructive solid geometry (CSG)



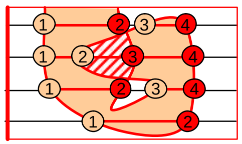
Collision queries

Vertex-in-volume test | **Explicit intersection volume**



Self-collision query

- check for incorrect ordering of front and back faces
- if front and back faces do not alternate -> self collision

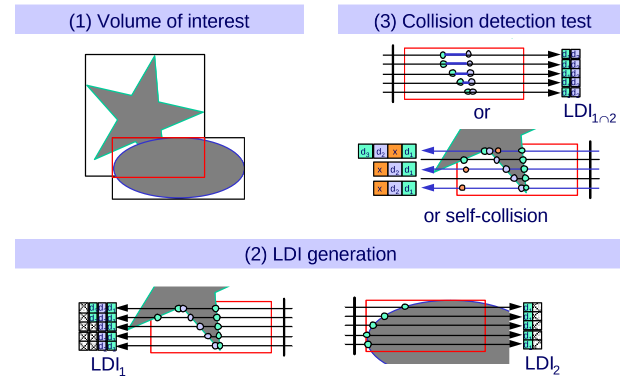


VoI

1	2	3	4
1	2	3	4
1	2	3	4
1	2		

LDI

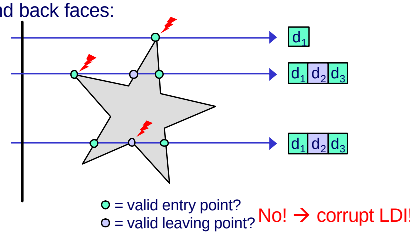
Algorithm Summary



Problems

- object can not be rendered to shadow map (see differences to depth buffer) → additional copy process necessary
- limited precision of depth buffer leads to singularities near edges between front and back faces:

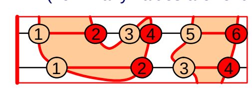
example:



→ handle front and back faces in separate passes

Unordered LDI Generation

- alternative method for LDI generation
- GPU generates unsorted LDI
  - fragments are rendered in the same order in each rendering pass
  - stencil buffer is used to get n-th value in the n-th pass
- CPU generates ordered LDI
  - depth complexity is known for each fragment (how many values are rendered per pixel)



VoI

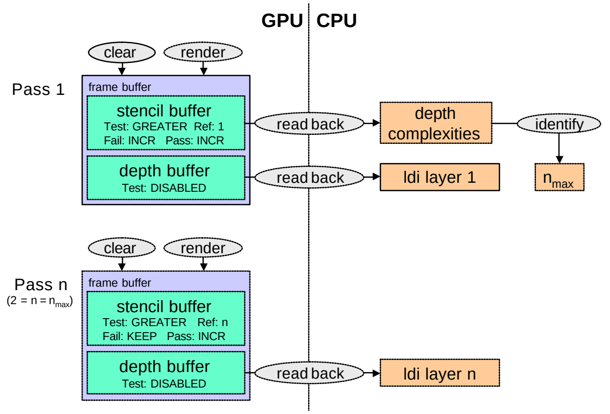
5	3	2	1	4	6
4	1	3	2	2	2

unsorted LDI (GPU)

1	2	3	4	5	6
1	2	3	4	2	2

sorted LDI (CPU)

Unordered LDI Generation



Limitations

- performance is dependent on:
  - depth complexity of objects in volume of interest
  - read back delay for simple objects
  - rendering speed for complex objects
- requires graphics hardware

Ordered LDI Generation on CPU

Motivation

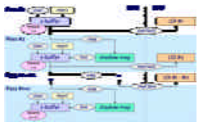
- buffer read-back from GPU can be performance bottleneck
- GPU requires multiple passes
- CPU can store fragments directly into LDI

Simplified software-renderer

- rasterization of triangle meshes
- frustum culling
- face clipping
- orthogonal projection

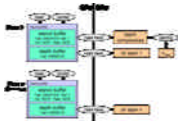
LDI Generation - Summary

Ordered LDI (GPU)



- n+1 passes
- complex setup
- two depth tests
- shadow map
- OpenGL extensions

Unordered LDI (GPU)



- n passes
- simple setup
- no depth test
- stencil buffer
- plain OpenGL 1.4

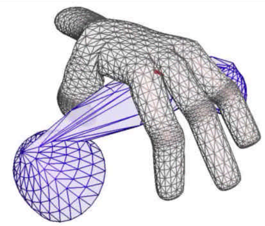
Ordered LDI (CPU)

rasterize

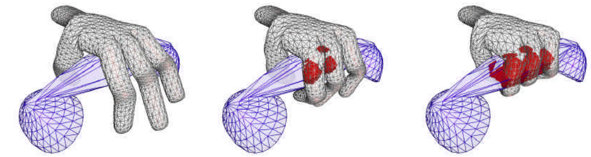
- 1 pass
- simple setup
- no depth test

Performance - Intersection Volume

- hand with 4800 faces
- phone with 900 faces
- two LDIs
- intersection volume
  for collision detection
- analysis of front / back face
  ordering for self-collision



Performance – Intersection Volume



method	collision min / max	self collision min / max	overall min / max
ordered (GPU)	28 / 37	40 / 54	68 / 91
unordered (GPU, CPU)	9 / 12	12 / 18	21 / 30
software (CPU)	3 / 4	5 / 7	8 / 11

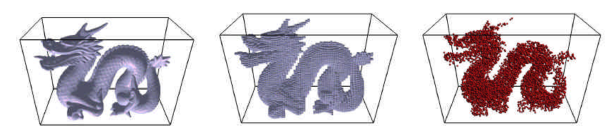
3 GHz Pentium 4, GeForce FX Ultra 5800

hand with 4800 faces
phone with 900 faces
measurements in ms

Performance – Vertex-in-Volume

- santa with 10000 faces
- 20000 particles
- one LDI
- test vertices against
  inside regions of the LDI

Performance – Vertex-in-Volume



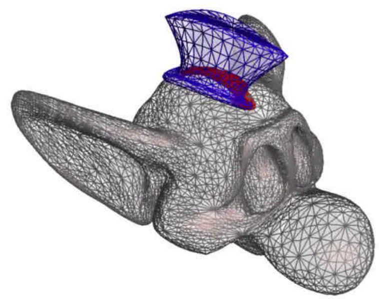
method	520k faces 100k particles	150k faces 30k particles	50k faces 10k particles
ordered (GPU)	450	160	50
unordered (GPU, CPU)	225	75	25
software (CPU)	400	105	35

3 GHz Pentium 4, GeForce FX Ultra 5800

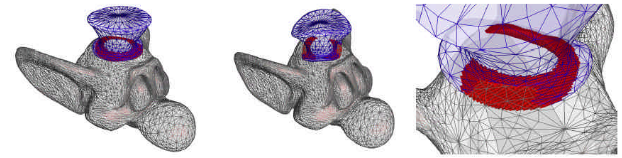
LDI resolution 64 x 64
measurements in ms

Performance – LDI resolution

- mouse with 15000 faces
- hat with 1500 faces
- two LDIs
- intersection volume for collision detection



Performance – LDI resolution



method	32 x32	64 x 64	128 x128
ordered (GPU)	24	26	51
unordered (GPU, CPU)	8	9	17
software (CPU)	2	3	6

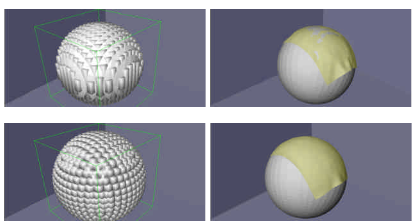
3 GHz Pentium 4, GeForce FX Ultra 5800

mouse with 15000 faces
hat with 1500 faces
measurements in ms

Applications – Cloth Modeling

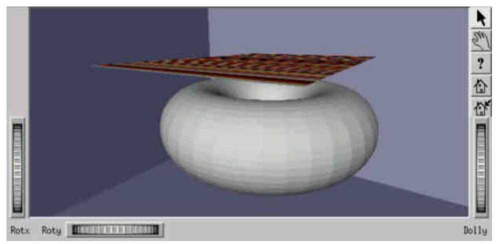
LDI

3 orthogonal dilated LDIs



Real-Time Cloth Simulation with Collision Handling

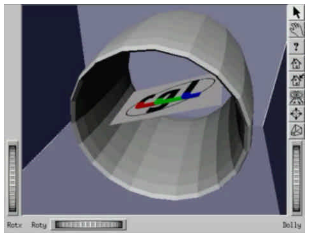
real-time movie
3GHz Pentium 4



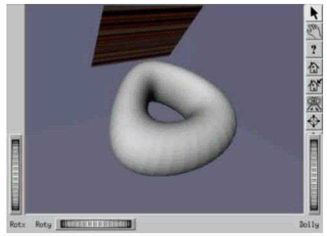
stable collision handling

Real-Time Cloth Simulation with Collision Handling

real-time movies
3GHz Pentium 4



concave transforming object



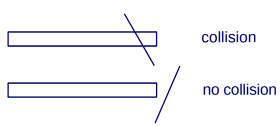
concave deforming object

Summary

- image-space technique
- detection of collisions and self-collisions
- handling of rigid and deformable closed meshes
- no pre-processing
- CPU: 5000 / 1000 faces at 100 Hz
- GPU: 520000 faces / 100000 particles at 4 Hz
- application to cloth simulation
- limitations
  - closed meshes
  - accuracy
  - collision information for collision response

Image-Space Collision Detection with a Box [Lombardo 1998]

- collision detection of a surgical tool and an anatomical structure
- tool is modeled as a box
- viewing volume of a camera is specified based on this box (near, far, left, right, top, bottom)
- anatomical structure is rendered in terms of this camera
- if something has been rendered → collision
- if nothing has been rendered → no collision



Intersection Detection for Deformable Objects

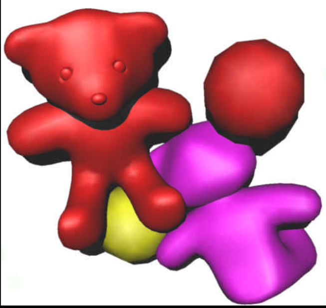
Bounding Volume Hierarchies
- efficient or lazy update of BV hierarchies
- hierarchy update is essential for performance

Spatial Partitioning with Hashing
- detects self-collisions
- appropriate for deformable objects or many objects

Spatial Partitioning with Graphics Hardware
- rendering of objects provides spatial partitioning
- rendering result can be employed for collision detection
- LDIs can be used to approximately represent objects for further processing

References

- M. Shinya, M. Forgue, "Interference Detection through rasterization," *Journal of Visualization and Computer Animation*, vol. 2, pp. 132-134, 1991.
- K. Myszkowski, O. Okunev, T. Kunii, "Fast collision detection between complex solids using rasterizing graphics hardware," The Visual Computer, vol. 11, no. 9, pp. 497-512, 1995.
- J. C. Lombardo, M.-P. Cani, F. Neyret, "Real-time Collision Detection for Virtual Surgery," *Proc. of Comp. Anim*., pp. 82-91, 1999.
- G. Baciu, S. K. Wong "Image-Based Techniques in a Hybrid Collision Detector," *IEEE Trans on Visualization and Computer Graphics*, Jan 2002.
- D. Knott, D. Pai: "Cinder: Collision and interference detection in real-time using graphics hardware," *Proc. Graphics Interface, 2003*.
- B. Heidelberger, M. Teschner, M. Gross**,** *"Volumetric Collision Detection for Deformable Objects," Proc. VMV'03*, pp. 461-468, 2003.
- B. Heidelberger, M. Teschner, M. Gross, "Detection of Collisions and Self-collisions Using Image-space Techniques," *Proc. WSCG'04*, pp. 145-152, 2004.



Proximity Queries

Simulation in Computer Graphics
University of Freiburg

WS 04/05

Acknowledgements

- parts of this slide set are courtesy of
  Bruno Heidelberger, ETH Zurich
- parts of this slide set are based on
  G. van den Bergen, “Collision Detection
  in Interactive 3D Environments,”
  Elsevier, Amsterdam, ISBN: 1-55860-801-X, 2004.

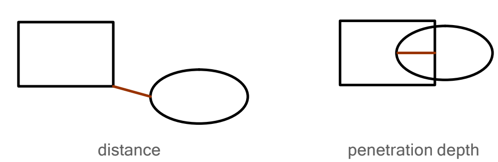
Outline

- introduction
- Minkowski sum
- distance computation
  Gilbert-Johnson-Keerthi algorithm (GJK)
- penetration depth computation
  expanding-polytope algorithm (EPA)
- approximate distance
- approximate consistent penetration depth
- demos

University of Freiburg - Institute of Computer Science - Computer Graphics Laboratory

Proximity Query

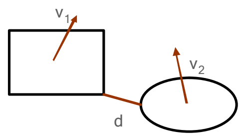
- for a pair of objects
  - compute their distance
    (find a pair of closest points)
  - compute their penetration depth
    (minimal translation to separate two interfering objects)

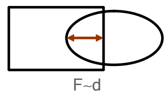


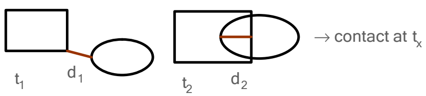
University of Freiburg - Institute of Computer Science - Computer Graphics Laboratory

Application

- distance
  - collision candidates
  - continuous collision detection
- penetration depth
  - penalty-based collision response
  - computation of time of contact





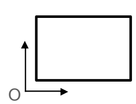


Outline

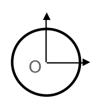
- introduction
- Minkowski sum
- distance computation
  Gilbert-Johnson-Keerthi algorithm (GJK)
- penetration depth computation
  expanding-polytope algorithm (EPA)
- approximate distance
- approximate consistent penetration depth
- demos

Minkowski Addition

- A

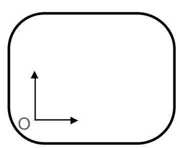


B



- $A + B = \{ x + y : x \in A, y \in B \}$

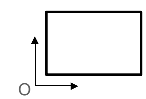
  $(A + t_1) + (B + t_2) = (A + B) + t_1 + t_2$



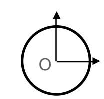
- representation of swept objects

Configuration Space Obstacle

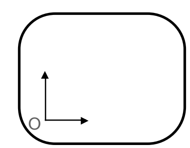
- A



B



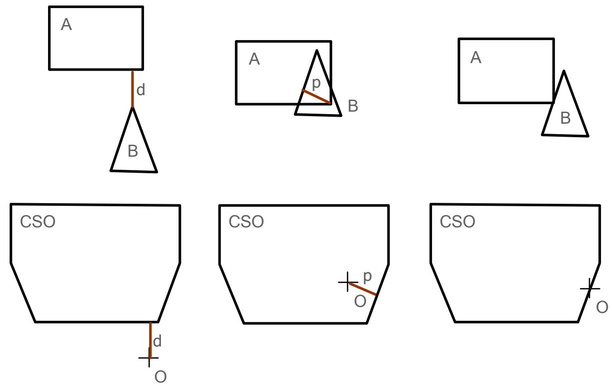
- $CSO(A,B) = A - B = A + (-B) = \{ x - y : x \in A, y \in B \}$
- to realize A-B,
  the reflection of B is added to A



CSO and Proximity Queries

- iff A and B intersect, they have a common point $x_1 = y_1$ with $x_1 - y_1 = O$
- $\rightarrow O \in CSO(A,B)$ iff A and B intersect
- d (A,B) distance between A and B
  $d(A,B) = \min \{ || x - y || : x \in A, y \in B \}$
- p (A,B) penetration depth of A and B
  $p(A,B) = \inf \{ || x || : x \notin CSO(A,B) \}$

Proximity Queries - Examples



Convex Objects

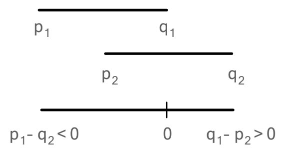
- if A and B are convex, then A+B and CSO(A,B) are convex
- proof:
  - let $w_1 = x_1 + y_1, w_2 = x_2 + y_2, x_1, x_2 \in A, y_1, y_2 \in B, w_1, w_2 \in A + B$
  - A+B is convex iff $\lambda_1 w_1 + \lambda_2 w_2 \in A + B, \lambda_1 + \lambda_2 = 1, \lambda_1, \lambda_2 \geq 0$
  - A is convex $\Rightarrow \lambda_1 x_1 + \lambda_2 x_2 \in A$
  - B is convex $\Rightarrow \lambda_1 y_1 + \lambda_2 y_2 \in B$
  - $\lambda_1 x_1 + \lambda_2 x_2 + \lambda_1 y_1 + \lambda_2 y_2 = \lambda_1 (x_1 + y_1) + \lambda_2 (x_2 + y_2) = \lambda_1 w_1 + \lambda_2 w_2$
  - $\Rightarrow \lambda_1 w_1 + \lambda_2 w_2 \in A + B$
  - $\Rightarrow$ A+B is convex
- important for computing proximity queries on CSOs for convex objects

Convex Polytopes

- A and B are polytopes, e. g. closed triangulated surfaces
- conv (A) - convex hull of A
- vert (A) - set of vertices of A
- A + B = conv (vert (A) + vert (B))
- computing the convex hull for all pair wise sums of vertices of A and B gives the Minkowski sum of A and B
- important for computing A + B for convex polytopes

Proximity Queries - AABBs

- axis-aligned boxes $A = [\, p_1, q_1 \,]$, $B = [\, p_2, q_2 \,]$
- CSO (A, B) $= [\, p_1, q_1 \,] - [\, p_2, q_2 \,] = [\, p_1 - q_2, q_1 - p_2 \,]$
- A and B intersect iff $O \in [\, p_1 - q_2, q_1 - p_2 \,]$
- intersecting AABBs in 1D



Proximity Queries - AABBs

- axis-aligned boxes
  $A = [\, c_1 - h_1, c_1 + h_1]$, $B = [\, c_2 - h_2, c_2 + h_2]$, $h_1, h_2 > O$
- CSO (A, B) $= [\, c_1 - c_2 - (h_1 + h_2), c_1 - c_2 + (h_1 + h_2) \,]$
- $O \in$ CSO (A, B) iff $|c_1 - c_2| < h_1 + h_2$
  (see BVH slides)
- intersection test for spheres can be derived
  in a similar way

Summary

- Minkowski sum or configuration space obstacle CSO
  can be used for proximity queries
- if origin is not contained in CSO,
  then the distance of two objects is given by
  the distance of the CSO to the origin
- if origin is contained in CSO, the penetration depth
  is given by the distance of the CSO to the origin
- useful characteristics for CSO of convex polytopes
- intersection tests for AABBs and other basic primitives
  can be derived from CSO

Outline

- introduction
- Minkowski sum
- distance computation
  Gilbert-Johnson-Keerthi algorithm (GJK)
- penetration depth computation
  expanding-polytope algorithm (EPA)
- approximate distance
- approximate consistent penetration depth
- demos

Overview

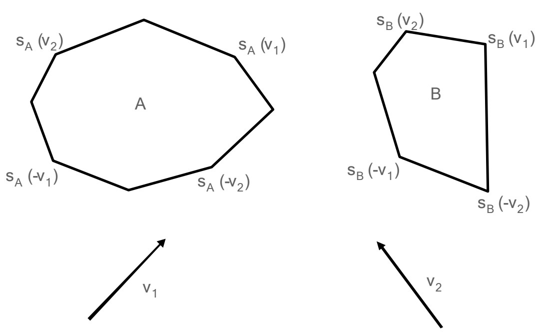
- for a given convex polytope C with $O \notin C$,
  GJK computes the point v (C) closest to the origin O
- $|| v(C) || = \min (|| x || : x \in C)$
- iff C = CSO (A, B), then GJK computes the distance
  d (A, B) of two non-intersecting convex objects A and B
- $d(A, B) = || v(CSO(A, B)) ||$

Support Mapping

- A support mapping of a polytope A is a function s_A
  that maps a vector v to a vertex of A.
- $s_A(v) \in vert(A)$ with $v \cdot s_A(v) = \max (v \cdot a : a \in vert(A))$
- The vertex $s_A(v)$ is
  the support point of A with respect to v.

University of Freiburg - Institute of Computer Science - Computer Graphics Laboratory

Support Mapping - Example



University of Freiburg - Institute of Computer Science - Computer Graphics Laboratory

Support Mapping for Convex Polytopes

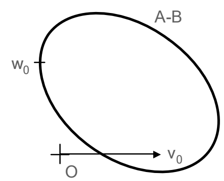
- represent the convex polytope as an adjacency graph
- start with an initial guess
- "climb the hill" by searching the adjacency graph
  for better solutions $\Rightarrow$ hill climbing

- p = cached support vertex
- repeat
  - optimal = true
  - for $q \in adj(p)$ do
    - if $v \cdot q > v \cdot p$ then { p = q, optimal = false }
- until optimal

University of Freiburg - Institute of Computer Science - Computer Graphics Laboratory

GJK Initialization – Step 0

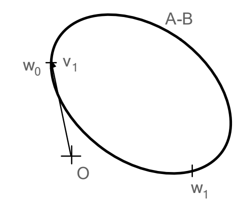
- iterative approximation of d (A, B)
- GJK starts with an arbitrary $v_0 \in A - B$ and a set of vertices $W_0 = \varnothing$



- $w_0 = s_{A-B} (-v_0)$

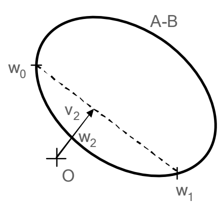
Step 1

- $v_1 = v (\text{conv} (W_0 \cup \{w_0\})) = v (\text{conv} (w_0))$
- $w_1 = s_{A-B} (-v_1)$
- W_1 = "smallest" X with $X \subseteq W_0 \cup \{w_0\}$ such that $v_1 \in \text{conv}(X)$
- $W_1 = \{w_0\}$



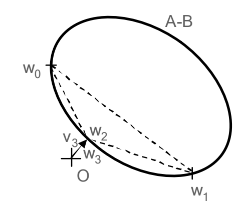
Step 2

- $v_2 = v (\text{conv} (W_1 \cup \{w_1\})) = v (\text{conv} (w_0, w_1))$
- $w_2 = s_{A-B} (-v_2)$
- W_2 = "smallest" X with $X \subseteq W_1 \cup \{w_1\}$ such that $v_2 \in \text{conv}(X)$
- $W_2 = \{ w_0, w_1 \}$



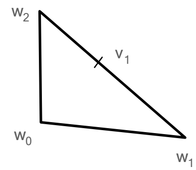
Step 3

- $v_3 = v (\text{conv} (W_2 \cup \{w_2\})) = v (\text{conv} (w_0, w_1, w_2))$
- $w_3 = s_{A-B} (-v_3)$
- W_3 = "smallest" X with $X \subseteq W_2 \cup \{w_2\}$ such that $v_3 \in \text{conv}(X)$
- $W_3 = \{ w_2 \}$



"smallest" X

- $v_1 = v$ (conv (w_0, w_1, w_2)) $X = \{ w_0, w_1, w_2 \}$
- $v_1 = \lambda_0 w_0 + \lambda_1 w_1 + \lambda_2 w_2$ with $\lambda_0+\lambda_1+\lambda_2=1$, $\lambda_0,\lambda_1,\lambda_2 \geq 0$
- if $\lambda_i = 0$ then the corresponding w_i can be removed from X such that $v_1 = v$ (conv (X))
- example:
- $v_1 = \lambda_0 w_1 + \lambda_1 w_2$
- $\Rightarrow v_1 = v$ (conv (w_1, w_2))
- $\Rightarrow X = \{ w_1, w_2 \}$



GJK Algorithm

- v = arbitrary point in A - B
- $W = \emptyset$
- $w = s_{A-B}$ (-v)
- while v not close enough to v (A-B)
  - v = v (conv ($W \cup \{w\}$))
  - W = smallest $X \subseteq W \cup \{w\}$ such that $v \in$ conv (X)
  - $w = s_{A-B}$ (-v)
- return ||v||

University of Freiburg - Institute of Computer Science - Computer Graphics Laboratory

Convergence and Termination

- $|| v_{k+1} || \leq || v_k ||$
- if $|| v_{k+1} || = || v_k ||$ then $v_k = v$ (A-B)
- for polytopes, GJK computes $v_k = v$ (A-B) in a finite number of iterations
- for non-polytopes, the error of $|| v_k ||$ is bound by $|| v_k - v (A\text{-}B) ||^2 \leq || v_k ||^2 - v_k \cdot w_k$

University of Freiburg - Institute of Computer Science - Computer Graphics Laboratory

Summary

- GJK computes the distance of two non-intersecting objects
- iterative process
- main loop performs three steps on a simplex
  - computation of the distance of the simplex to the origin
  - support mapping based on this distance
  - adaptation of the simplex based on the support point
- GJK converges to the correct solution
- GJK computes the distance in a finite number of iterations for polytopes

University of Freiburg - Institute of Computer Science - Computer Graphics Laboratory

Outline

- introduction
- Minkowski sum
- distance computation
  Gilbert-Johnson-Keerthi algorithm (GJK)
- penetration depth computation
  expanding-polytope algorithm (EPA)
- approximate distance
- approximate consistent penetration depth
- demos

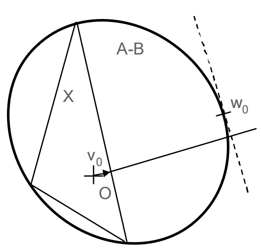
Introduction

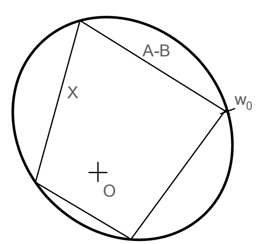
- EPA computes the penetration depth of two objects
- iterative process
- works with an CSO that contains the origin
- starts with a simplex (triangle in 2D, tetrahedron in 3D) that contains the origin and whose vertices are on the boundary of the CSO
- the initial simplex is subdivided (expanded) by EPA to approximate the CSO
- the distance of the expanded polytope to the origin corresponds to the penetration depth

University of Freiburg - Institute of Computer Science - Computer Graphics Laboratory

Step 0

- $v_0 = v(X)$
- $w_0 = s_{A-B}(v_0)$
- expand X such that it contains w_0

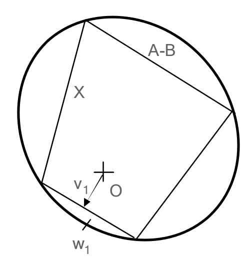


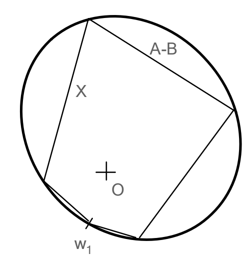


University of Freiburg - Institute of Computer Science - Computer Graphics Laboratory

Step 1

- $v_1 = v(X)$
- $w_1 = s_{A-B}(v_1)$
- expand X such that it contains w_1

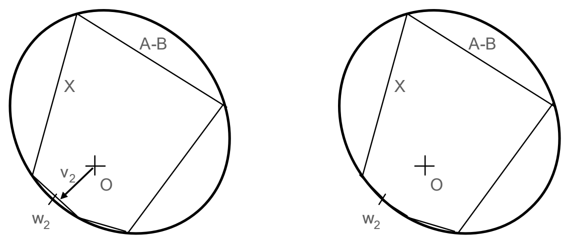




University of Freiburg - Institute of Computer Science - Computer Graphics Laboratory

Step 2

- $v_2 = v(X)$
- $w_2 = s_{A-B}(v_2)$
- expand X such that it contains w_2



Convergence and Termination

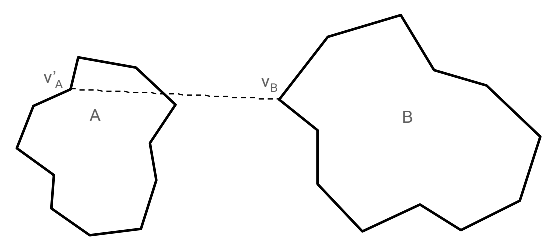
- $\| v_{k+1} \| \geq \| v_k \|$
- for polytopes, EPA computes $v_k = v(A-B)$ in a finite number of iterations

Outline

- introduction
- Minkowski sum
- distance computation
  Gilbert-Johnson-Keerthi algorithm (GJK)
- penetration depth computation
  expanding-polytope algorithm (EPA)
- approximate distance
- approximate consistent penetration depth
- demos

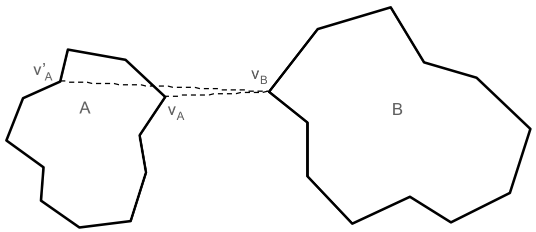
Approximate Distance – Step 1

- two polytopes A and B
- start with an arbitrary vertex v'_A with $v'_A \in$ vert (A)
- compute nearest vertex v_B with $v_B \in$ vert (B)



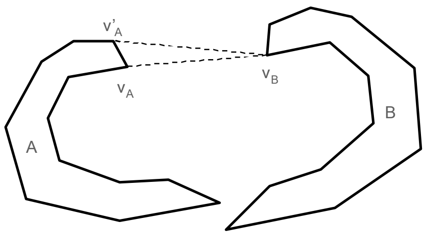
Approximate Distance – Step 2

- compute nearest vertex $v_A \in$ vert (A) with respect to v_B
- $|| v_A - v_B ||$ is the approximate distance of A and B



Characteristics

- better approximation for larger distances and convex objects
- bad approximation in case of concave objects

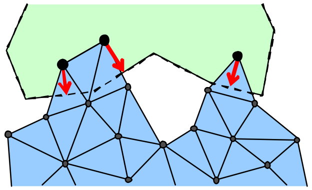


Outline

- introduction
- Minkowski sum
- distance computation Gilbert-Johnson-Keerthi algorithm (GJK)
- penetration depth computation expanding-polytope algorithm (EPA)
- approximate distance
- approximate consistent penetration depth
- demos

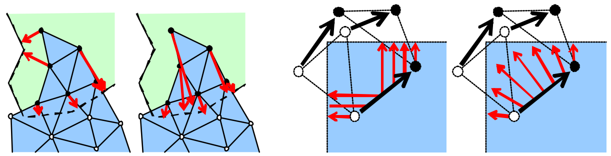
Motivation

- compute consistent penetration depth information for all intersecting points of a tetrahedral mesh
- can be used to compute penalty forces which provide realistic collision response for deformable tetrahedral meshes



Challenges

- inconsistent penetration depth information due to discrete simulation steps and object discretization

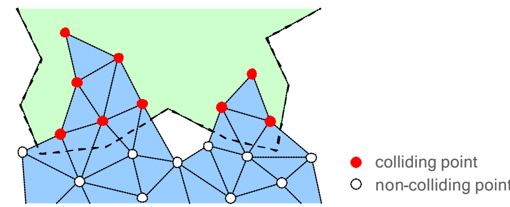


inconsistent consistent inconsistent consistent

- inconsistent penetration depth results in oscillation artifacts or non-realistic collision response

Algorithm – Stage 1

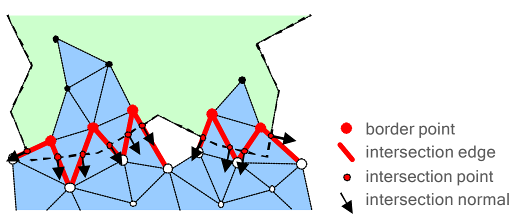
- object points are classified as colliding or non-colloding points → slides on spatial hashing



University of Freiburg - Institute of Computer Science - Computer Graphics Laboratory

Algorithm – Stage 2

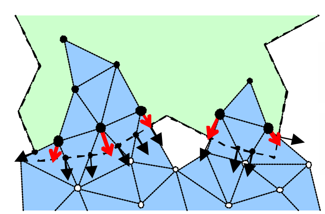
- border points, intersecting edges, and intersection points are detected → extension of spatial hashing



University of Freiburg - Institute of Computer Science - Computer Graphics Laboratory

Algorithm – Stage 3

- penetration depth d(p) of a border point p is approximated using the adjacent intersection points x_i and normals n_i



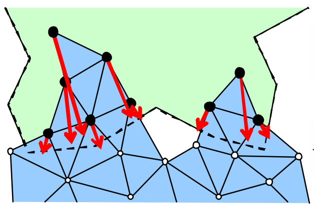
$$\omega(\mathbf{x}_i, \mathbf{p}) = \frac{1}{\|\mathbf{x}_i - \mathbf{p}\|^2}$$

$$d(\mathbf{p}) = \frac{\sum_{i=1}^{k} (\omega(\mathbf{x}_i, \mathbf{p}) \cdot (\mathbf{x}_i - \mathbf{p}) \cdot \mathbf{n}_i)}{\sum_{i=1}^{k} \omega(\mathbf{x}_i, \mathbf{p})}$$

University of Freiburg - Institute of Computer Science - Computer Graphics Laboratory

Algorithm – Stage 4

- consistent penetration depth information at points p_j is propagated to other colliding points p

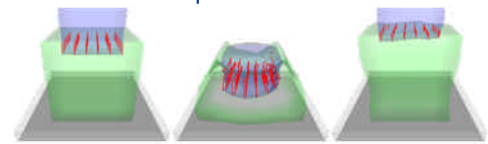


$$\mu(\mathbf{p}_j,\mathbf{p}) = \frac{1}{\|\mathbf{p}_j - \mathbf{p}\|^2}$$

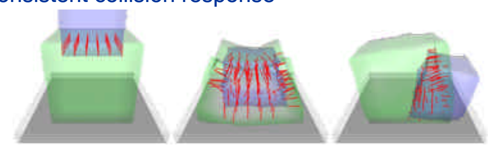
$$\frac{\sum_{j=1}^{l}(\mu(\mathbf{p}_j,\mathbf{p}) \cdot ((\mathbf{p}_j - \mathbf{p}) \cdot \mathbf{r}(\mathbf{p}_j) + d(\mathbf{p}_j)))}{\sum_{j=1}^{l}\mu(\mathbf{p}_j,\mathbf{p})}$$

Results

- consistent collision response



- inconsistent collision response



University of Freiburg - Institute of Computer Science - Computer Graphics Laboratory

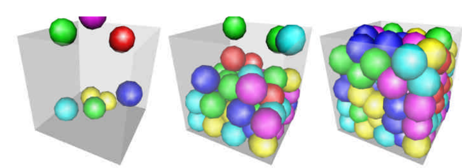
Results - Video

Consistent Penetration
Depth Estimation
for Deformable
Collision Response

University of Freiburg - Institute of Computer Science - Computer Graphics Laboratory

Summary

- consistent penetration depth information in case of
  - discrete object representation
  - discrete time simulation
- addresses the problem of discontinuities in magnitude and direction of the penetration depth
- provides realistic penalty-based collision response



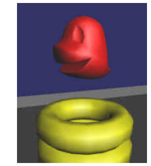
University of Freiburg - Institute of Computer Science - Computer Graphics Laboratory

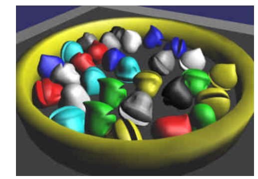
Outline

- introduction
- Minkowski sum
- distance computation
  Gilbert-Johnson-Keerthi algorithm (GJK)
- penetration depth computation
  expanding-polytope algorithm (EPA)
- approximate distance
- approximate consistent penetration depth
- demos

Interacting Deformable Objects

- deformable modeling based on constraints
- collision detection based on spatial hashing
- collision response based on consistent
  penetration depth computation





References

- E. G. Gilbert, D. W. Johnson, S. S. Keerthi, “A Fast Procedure for Computing the Distance Between Complex Objects in Three-Dimensional Space,” IEEE Journal of Robotics and Automation, vol. 4, no. 2, pp. 193-203, 1988.
- G. van den Bergen, “Collision Detection in Interactive 3D Environments,” Elsevier, Amsterdam, ISBN: 1-55860-801-X, 2004.
- B. Heidelberger, M. Teschner et al., “Consistent Penetration Depth Estimation for Deformable Collision Response,” Proc. VMV, Stanford, USA, 2004.

Fast Collision Detection among Deformable Objects using Graphics Processors

Naga K. Govindaraju   Dinesh Manocha

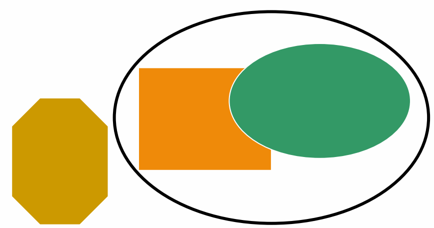
Outline

- Overview
- Interactive Collision Detection
- Conclusions and Future Work

Collision Detection

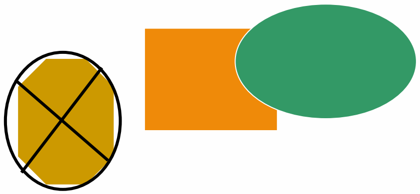
- Well studied
  - Computer graphics, computational geometry etc.
- Widely used in games, simulations, virtual reality applications
  - Often a computational bottleneck

Interactive Collision Detection



Interactive Collision Detection

- Visibility to reduce number of pair-wise overlap tests



Interactive Collision Detection

Graphics Processing Units (GPUs)

- Well-designed for visibility computations
  - Rasterization – image-space visibility
- Massively parallel
  - Render millions of polygons per second
  - Well suited for image-based algorithms
- High growth rate

Recent growth rate of Graphics Processing Units

Card	Million triangles/sec
Radeon 9700 Pro	325
GeForce FX 5800	350
Radeon 9800 XT	412
GeForce FX 5950	356
GeForce FX 6800	**600**

Graphics Processing Units (GPUs)

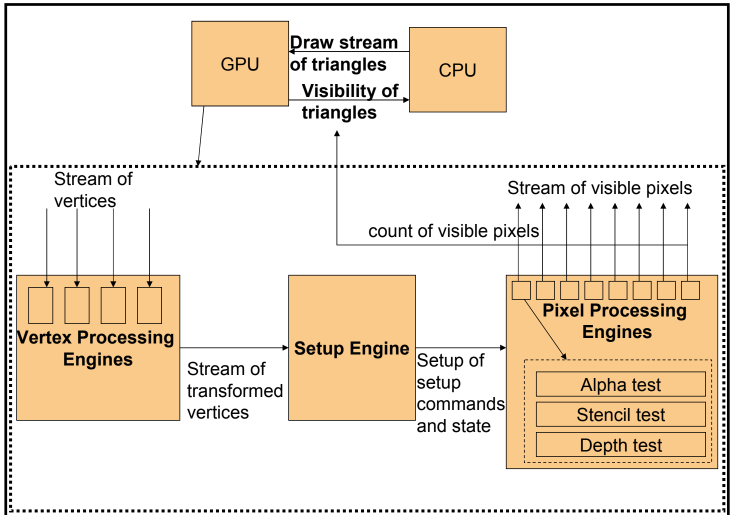
- Well-designed for visibility computations
  - Rasterization – image-space visibility
- Massively parallel
  - Render millions of polygons per second
  - Well suited for image-based algorithms
- High growth rate

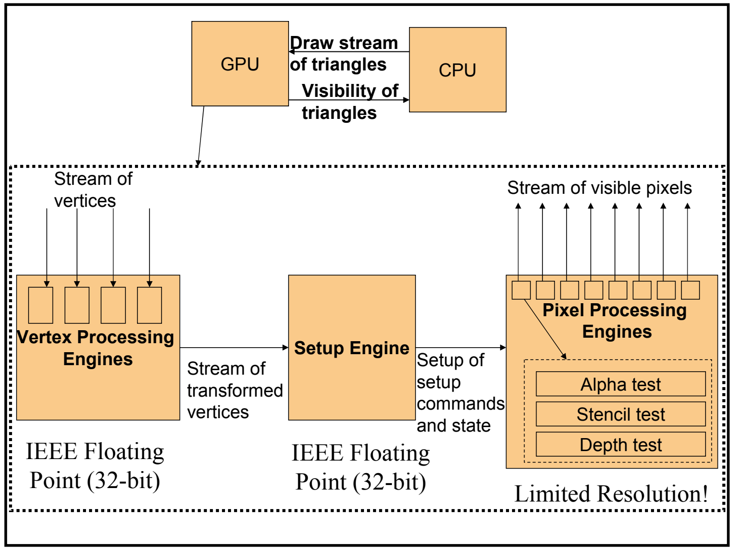
GPUs: Geometric Computations

- Used for geometric applications
  - Minkowski sums [Kim et al. 02]
  - CSG rendering [Goldfeather et al. 89, Rossignac et al. 90]
  - Voronoi computation [Hoff et al. 01, 02, Sud et al. 04]
  - Isosurface computation [Pascucci 04]
  - Map simplification [Mustafa et al. 01]

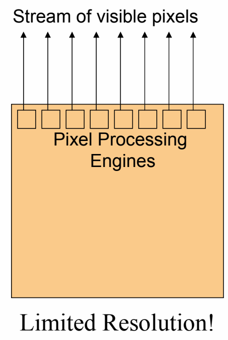
GPUs for Geometric Computations: Issues

- Precision
- Frame-buffer readbacks





Frame-Buffer Precision



Resolution along X, Y,Z

X – 12 bits fixed precision

Y – 12 bits fixed precision

Z – 24 bits fixed precision

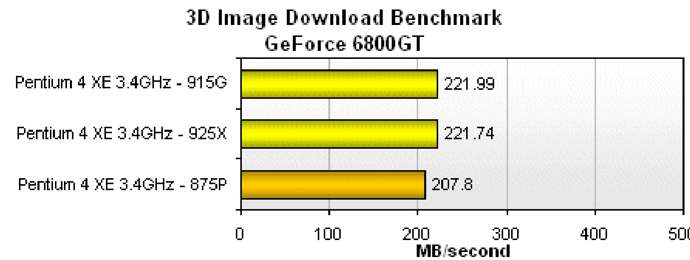
On CPU – 32-bit or 64-bit floating-point precision

Frame-Buffer Readback

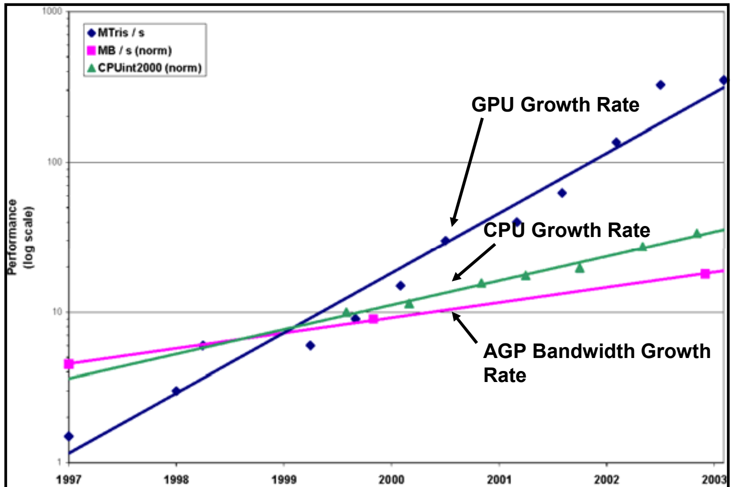
- Involve stalls
  - Affect throughput
- Slow!

Frame-Buffer Readback Performance

Data Courtesy: www.techreport.com
June 2004



Readback of 1Kx1K frame-buffer takes 18 ms over PCI-Express
Graphics driver – 61.45



Courtesy: Anselmo Lastra

Outline

- Overview
- Interactive Collision Detection
- Conclusions and Future Work

Features

Interactive collision detection between complex objects

- Large number of objects
- High primitive count
- Non-convex objects
- Open and closed objects

Non-rigid Motion

- Deformable objects
- Changing topology
- Self-collisions

Related Work

- Object-space techniques
- Image-space techniques

Object-Space Techniques

- Broad phase – Compute object pairs in close proximity
  - Spatial partitioning
  - Sweep-and-prune
- Narrow phase – Check each pair for exact collision detection
  - Convex objects
  - Spatial partitioning
  - Bounding volume hierarchies

Surveys in [Klosowski 1998, Redon et al. 2002, Lin and Manocha 2003]

Limitations of Object-Space Techniques

- Considerable pre-processing
- Hard to achieve real-time performance on complex deformable models

Collision Detection using Graphics Hardware

- Primitive rasterization – sorting in screen-space
  - Interference tests

Image-Space Techniques

Use of graphics hardware

- CSG rendering [Goldfeather et al. 1989, Rossignac et al. 1990]
- Interferences and cross-sections [Shinya and Forgue 1991 , Rossignac et al. 1992, Myszkowski 1995, Baciu et al. 1998]
- Minkowski sums [Kim et al. 2002]
- Cloth animation [Vassilev et al. 2001]
- Virtual Surgery [Lombardo et al. 1999]
- Proximity computation [Hoff et al. 2001, 2002]

Limitations of Image-Space Techniques

- Pairs of objects
- Stencil-based; limited to closed models
- Image precision
- Frame buffer readbacks

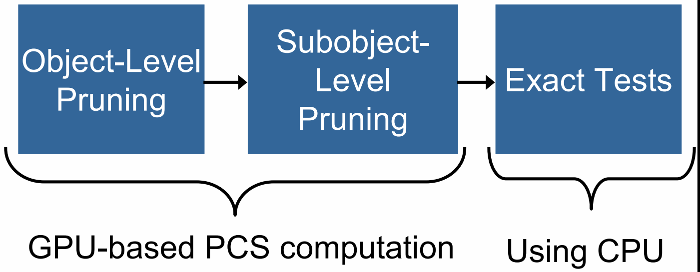
Collision Detection: Outline

- Overview
- Collision Detection: CULLIDE
- Inter- and Intra-Object Collision Detection: Quick-CULLIDE
- Reliable Collision Detection: FAR
- Analysis

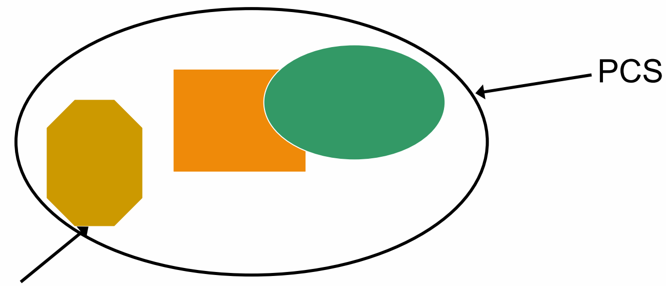
Overview

- Potentially Colliding Set (PCS) computation
- Exact collision tests on the PCS

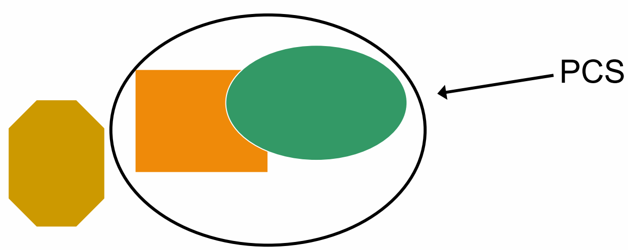
Algorithm



Potentially Colliding Set (PCS)



Potentially Colliding Set (PCS)



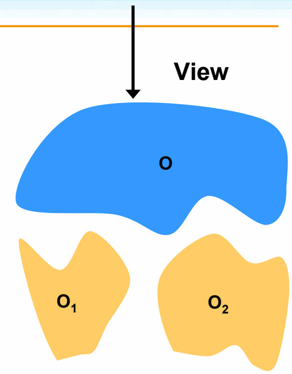
Algorithm

Visibility Computations

Lemma 1: *An object O does not collide with a set of objects S if O is fully visible with respect to S*

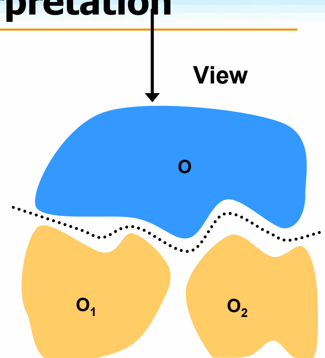
Visibility of Objects

- An object is fully visible if it is completely in front of the remaining objects



Visibility for Collisions: Geometric Interpretation

Sufficient but not a necessary condition for existence of separating surface with unit depth complexity



PCS Pruning

Lemma 2: *Given n objects $O_1, O_2, \ldots, O_n$, an object O_i does not belong to PCS if it does not collide with $O_1, \ldots, O_{i-1}, O_{i+1}, \ldots, O_n$*

- **Prune objects that do not collide**

PCS Pruning

$O_1 \quad O_2 \quad \ldots \quad O_{i-1} \; O_i \; O_{i+1} \quad \ldots \quad O_{n-1} \; O_n$

PCS Computation

- Each object tested against all objects but itself
- Naive algorithm is $O(n^2)$
- Linear time algorithm
  - Uses two pass rendering approach
  - Conservative solution

PCS Computation: First Pass

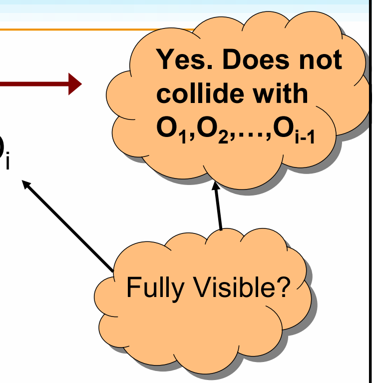
Render →

$O_1 \quad O_2 \quad \ldots \quad O_{i-1} \; O_i \; O_{i+1} \quad \ldots \quad O_{n-1} \; O_n$

PCS Computation: First Pass

Render →

$O_1 \quad O_2 \quad \ldots \quad O_{i-1} \; O_i$



PCS Computation: First Pass

Render →

O_1 O_2 … O_{i-1} O_i O_{i+1} … O_{n-1} O_n

Fully Visible?

PCS Computation: Second Pass

← Render

O_1 O_2 … O_{i-1} O_i O_{i+1} … O_{n-1} O_n

PCS Computation: Second Pass

← Render

Yes. Does not collide with $O_{i+1}, \ldots, O_{n-1}, O_n$

O_i O_{i+1} … O_{n-1} O_n

Fully Visible?

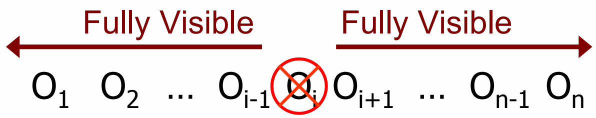
PCS Computation: Second Pass

← Render

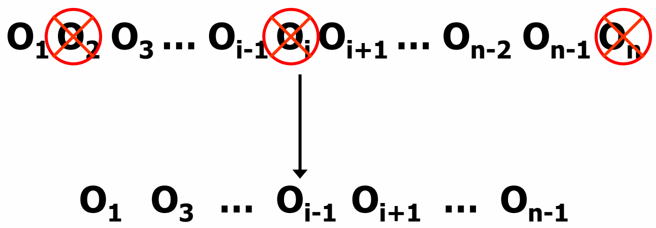
O_1 O_2 … O_{i-1} O_i O_{i+1} … O_{n-1} O_n

Fully Visible?

PCS Computation

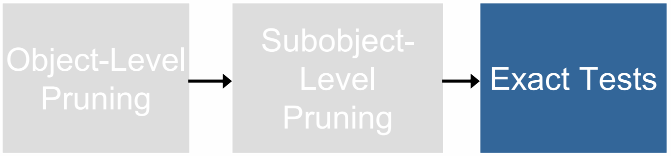


PCS Computation



Algorithm

CULLIDE Algorithm



Exact overlap tests using CPU

Full Visibility Queries on GPUs

- We require a query
  - Tests if a primitive is *fully visible* or not
- Current hardware supports occlusion queries
  - Test if only *part* of a primitive is *visible* or not
- Our solution
  - Change the sign of the depth function

Full Visibility Queries on GPUs

	Depth function	
	GEQUAL	LESS
All fragments	Pass	Fail
	Fail	Pass

Fail → Occlusion query

Pass → Query not supported

- **Examples - HP_Occlusion_test, NV_occlusion_query**

Bandwidth Analysis

- Read back only integer identifiers
  - Computation at high screen resolutions

Live Demo: CULLIDE

- Laptop
  - 1.6 GHz Pentium IV CPU
  - NVIDIA GeForce FX 700 GoGL
  - AGP 4X

Live Demo: CULLIDE

- Environment
  - Dragon – 250K polygons
  - Bunny – 35K polygons
- Average frame rate – 15 frames per second!

Interactive Collision Detection: Outline

- Overview
- Collision Detection: CULLIDE
- Inter- and Intra-Object Collision Detection: Quick-CULLIDE
- Reliable Collision Detection: FAR
- Analysis

Quick-CULLIDE

- Improved two-pass algorithm
- Utilize visibility relationships among objects across different views

Quick-CULLIDE: Visibility Sets

- Decompose PCS into four disjoint sets
  - FFV (First pass Fully Visible)
  - SFV (Second pass Fully Visible)
  - NFV (Not Fully Visible in either passes)
  - BFV (Both passes Fully Visible)
- Visibility sets have five interesting properties!

Visibility Sets: Properties

Lemma 1: FFV and SFV are collision-free sets

PCS Computation: First Pass

Render →

O_1 O_2 … O_{i-1} O_i … O_j … O_{n-1} O_n

PCS Computation: First Pass

Render →

O_1 O_2 … O_{i-1} O_i … O_j

Fully Visible

Visibility Sets: Properties

Lemma 2: It is sufficient to test visibility of objects in FFV in second pass only

PCS Computation: First Pass

O_1 O_2 … O_{i-1} O_i O_{i+1} … O_{n-1} O_n

PCS Computation: First Pass

Render →

O_1 O_2 … O_{i-1} O_i

PCS Computation: First Pass

Not Colliding ←

O_1 O_2 … O_{i-1} O_i O_{i+1} … O_{n-1} O_n

Collision tested
in Second pass

Visibility Sets: Properties

Lemma 3: It is sufficient to render objects in FFV in first pass only!

PCS Computation: First Pass

O_1 O_2 … O_{i-1} O_i O_{i+1} … O_{n-1} O_n

PCS Computation: First Pass

Render

O_1 O_2 … O_{i-1} O_i

PCS Computation

Render

Not Colliding

O_1 O_2 … O_{i-1} O_i O_{i+1} … O_{n-1} O_n

Visibility Sets: Properties

Lemma 4: It is sufficient to test the visibility of objects in SFV in first pass only!

Visibility Sets: Properties

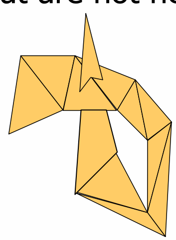
Lemma 5: It is sufficient to render objects in SFV in second pass only!

Quick-CULLIDE: Advantages

- Better culling efficiency
  - Lower depth complexity than CULLIDE
  - Always better than CULLIDE
- Faster computational performance
  - Lower number of visibility queries and rendering operations
- Can handle self-collisions

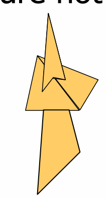
Self-Collisions: Definition

- Pairs of overlapping triangles in an object that are not neighboring



Self-Collisions: Definition

- Pairs of overlapping triangles in an object that are not neighboring



Self-Collisions

- Occur in most deformable simulations

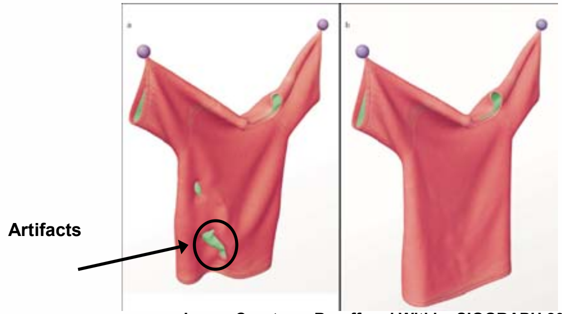
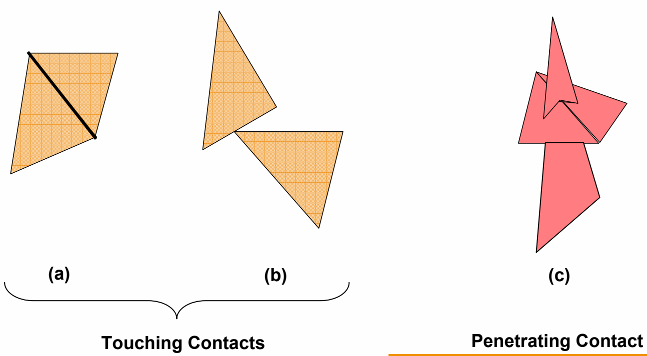


Image Courtesy: Baraff and Witkin, SIGGRAPH 2003

Our Solution

- Classification of contacts between triangles in an object
  - Touching contacts
  - Penetrating contacts

Contacts: Classification



Solution

- Ignore touching contacts
  - Consider only penetrating contacts
- Redefine fully visible
  - We pass a fragment when a touching contact occurs
  - Pass all fragments with depth ≤ corresponding depths in frame-buffer

Live Demo: Quick-CULLIDE

- Laptop
  - 1.6 GHz Pentium IV CPU
  - NVIDIA GeForce FX 700 GoGL
  - AGP 4X

Live Demo: Cloth Simulation

- Cloth – 20K triangles
- Average frame rate – 13 frames per second!

Interactive Collision Detection: Outline

- Overview
- Collision Detection: CULLIDE
- Self-Collision Detection: S-CULLIDE
- Reliable Collision Detection: FAR
- Analysis

Inaccuracies in GPU-Based Algorithms

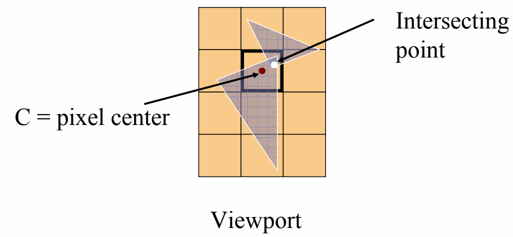
- Image sampling
- Depth buffer precision

Image Sampling

- Occurs when a primitive is nearly parallel to view direction

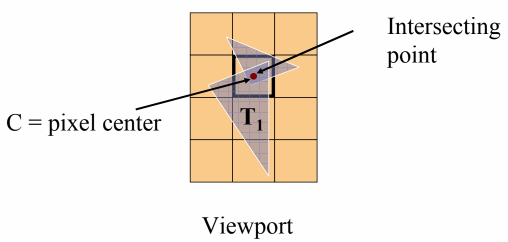
Image Sampling

- Primitives are rasterized but no intersecting points are sampled by hardware



Depth Buffer Precision

- Intersecting points are sampled but precision is not sufficient



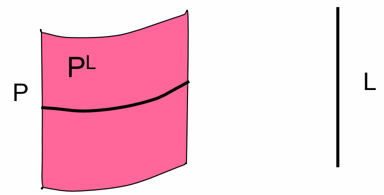
Our Solution

- Sufficiently fatten the triangles
  - Use Minkowski sums

Minkowski Sum $A^B = A \oplus B$

$= \{a + b: \ a \in A, b \in B\}$

Minkowski Sum: Example

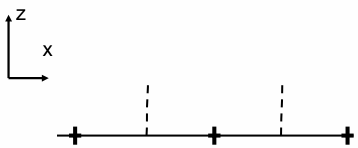


Reliability

Lemma 1: Under orthographic transformation O, the rasterization of Minkowski sum $Q^S = Q \oplus S$, where Q is a point in 3-D space that projects inside a pixel X and S is a sphere bounding a pixel centered at the origin, generates two samples for X that bound the depth value of Q.

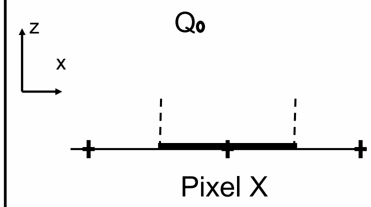
Reliability

***Under orthographic transformation O**, the rasterization of Minkowski sum $Q^S = Q \oplus S$, where Q is a point in 3-D space that projects inside a pixel X and S is a sphere centered at origin bounding a pixel, samples X with at least two fragments bounding the depth value of Q.*



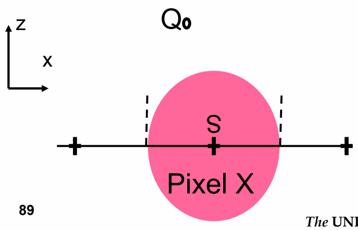
Reliability

*Under orthographic transformation O, the rasterization of Minkowski sum $Q^S = Q \oplus S$, where **Q is a point in 3-D space that projects inside a pixel X** and S is a sphere centered at origin bounding a pixel, samples X with at least two fragments bounding the depth value of Q.*



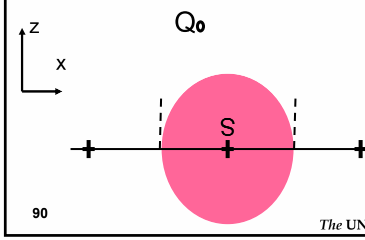
Reliability

Under orthographic transformation O, the rasterization of Minkowski sum $Q^S = Q \oplus S$, where Q is a point in 3-D space that projects inside a pixel X and ***S is a sphere centered at origin bounding a pixel,*** *samples X with at least two fragments bounding the depth value of Q.*



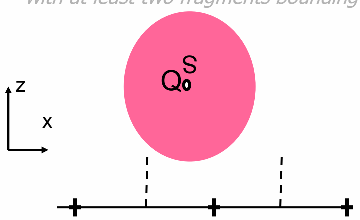
Reliability

Under orthographic transformation O, the rasterization of ***Minkowski sum $Q^S = Q \oplus S$,*** *where Q is a point in 3-D space that projects inside a pixel X and S is a sphere centered at origin bounding a pixel, samples X with at least two fragments bounding the depth value of Q.*



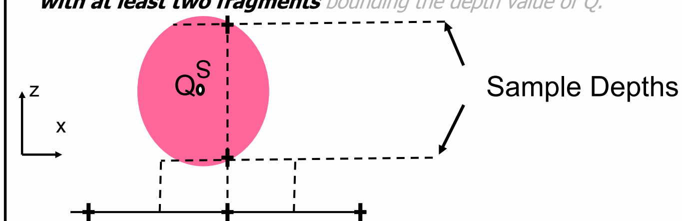
Reliability

Under orthographic transformation O, the rasterization of ***Minkowski sum $Q^S = Q \oplus S$,*** *where Q is a point in 3-D space that projects inside a pixel X and S is a sphere centered at origin bounding a pixel, samples X with at least two fragments bounding the depth value of Q.*



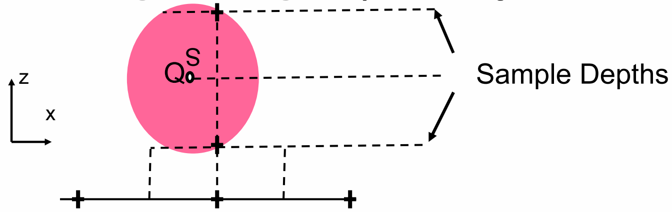
Reliability

Under orthographic transformation O, ***the rasterization of Minkowski sum $Q^S = Q \oplus S$,*** *where Q is a point in 3-D space that projects inside a pixel X and S is a sphere centered at origin bounding a pixel,* ***samples X with at least two fragments*** *bounding the depth value of Q.*



Reliability

*Under orthographic transformation O, the rasterization of Minkowski sum $Q^S = Q \oplus S$, where Q is a point in 3-D space that projects inside a pixel X and S is a sphere centered at origin bounding a pixel, samples X with at least two fragments bounding **the depth value of Q**.*



Reliability

Lemma 2: Given a primitive P and its Minkowski sum $P^S = P \oplus S$. Let X be a pixel partly or fully covered by the orthographic projection of P.

$P_x = \{p \in P$, p projects inside X$\}$,

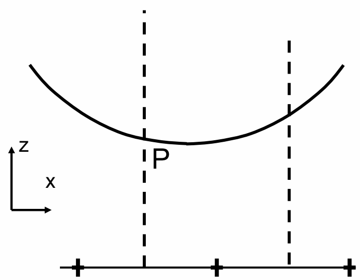
Min-Depth(P, X) = Minimum depth value in P_x

Max-Depth(P, X) =Maximum depth value in P_x.

The rasterization of P_x^S generates at least two fragments whose depth values bound both Min-Depth(P, X) and Max-Depth(P, X) for each pixel X.

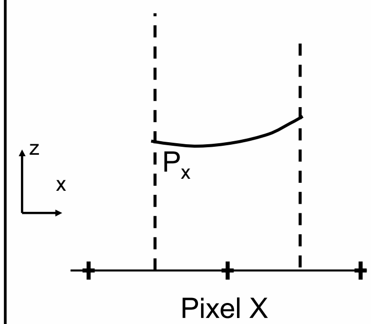
Reliability

Given a primitive P



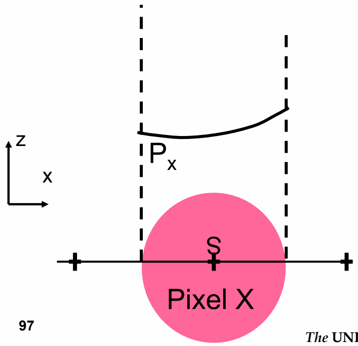
Reliability

P_x is the portion of P projecting inside pixel X



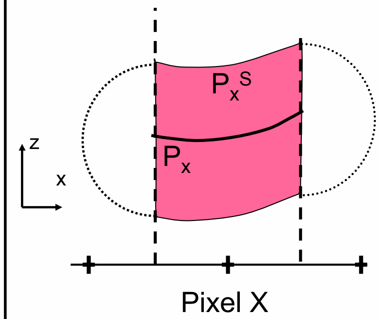
Reliability

S is a sphere centered at origin bounding pixel X



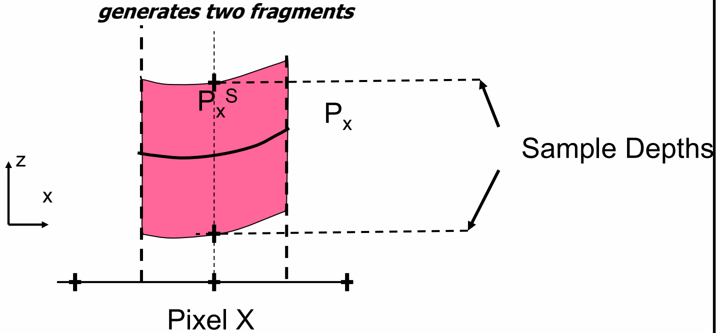
Reliability

If we compute Minkowski sum $P_x^S = P_x \oplus S$,



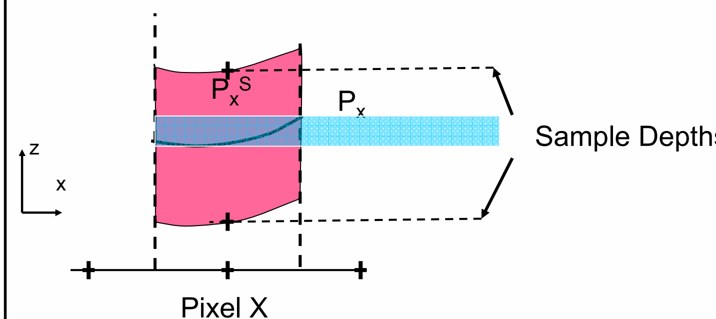
Reliability

then the rasterization of the Minkowski sum P_x^S generates two fragments



Reliability

and the fragments bound depth values in P_x

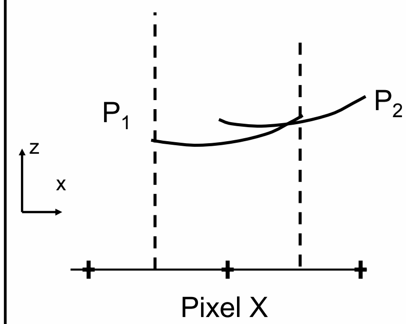


Reliability

Theorem 1: Given the Minkowski sum of two primitives with S, P_1^S and P_2^S. If P_1 and P_2 overlap, then a rasterization of their Minkowski sums under orthographic projection overlaps in the viewport.

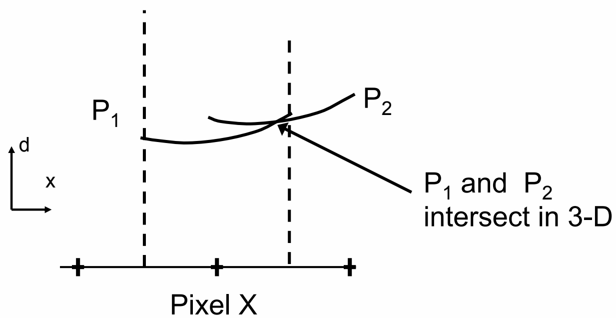
Reliability

Given two primitives P_1 and P_2



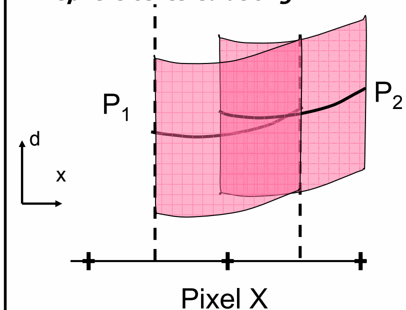
Reliability

If P_1 and P_2 intersect in 3-D,



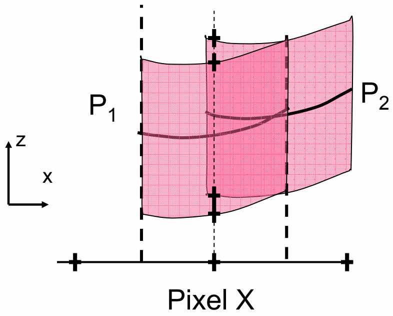
Reliability

and we compute their Minkowski sums with a pixel-sized sphere centered at origin



Reliability

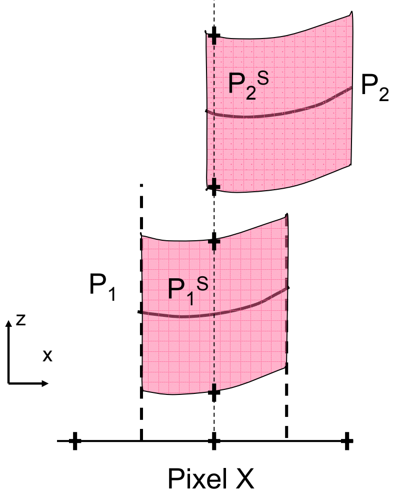
rasterization of the Minkowski sums overlap in image-space



Reliability

Corollary 1: Given the Minkowski sum of two primitives with B, P_1^S and P_2^S. If a rasterization of P_1^S and P_2^S under orthographic projection do not overlap in the viewport, then P_1 and P_2 do not overlap in 3-D.

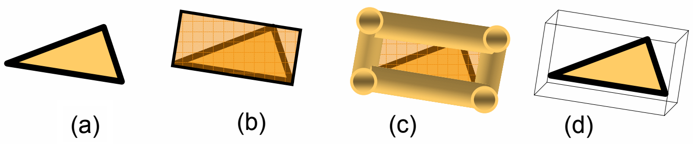
Useful in Collision Culling: apply fattened primitives P_1^S in CULLIDE



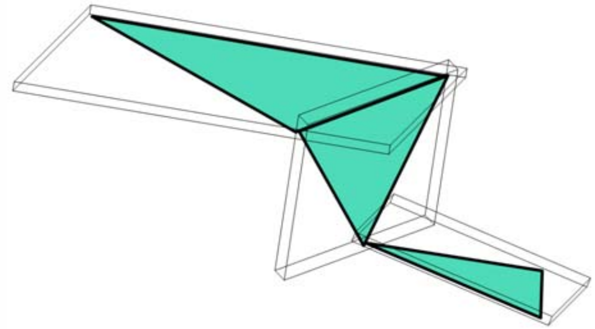
Bounding Offsets of a Triangle

- Exact Offsets
  - Three edge-aligned cylinders, three spheres, two triangles
  - Can be rendered using fragment programs
  - Expensive!
- Oriented Bounding Box (OBB)

OBB Construction



Union of OBBs



Live Demo: FAR

- Laptop
  - 1.6 GHz Pentium IV CPU
  - NVIDIA GeForce FX 700 GoGL
  - AGP 4X

Live Demo: FAR

- Environment
  - Tree – 4000 triangles
  - Leaf – 200 triangles, 200 leaves
  - Scene – 44K triangles
- Average frame rate – 15 frames per second!

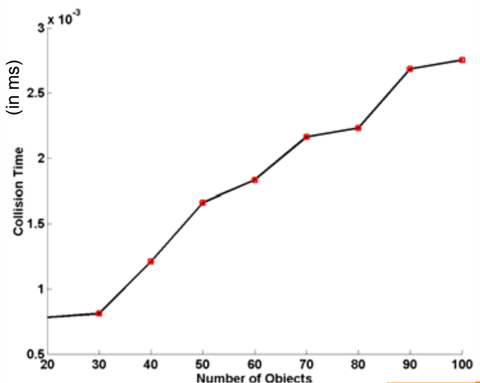
Interactive Collision Detection: Outline

- Overview
- Collision Detection: CULLIDE
- Self-Collision Detection: S-CULLIDE
- Reliable Collision Detection: FAR
- Analysis
  - Performance
  - Pruning efficiency
  - Precision

Analysis: Performance

- Based on pruning algorithm in CULLIDE
- Factors
  - Output size
  - Rasterization optimizations
  - Number of objects
  - Number of triangles per object
  - Image resolution

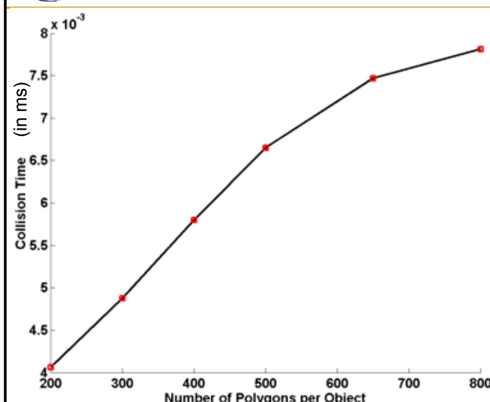
Analysis: Performance



Collision time
vs.
number of objects

NV30 GPU
Pentium IV 2GHz
CPU

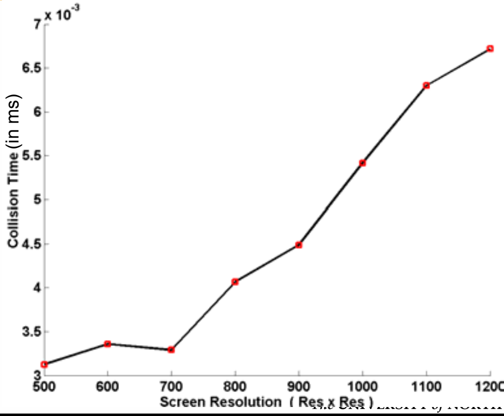
Analysis: Performance



Collision time
vs.
number of polygons

NV30 GPU
Pentium IV 2 GHz
CPU

Analysis: Performance



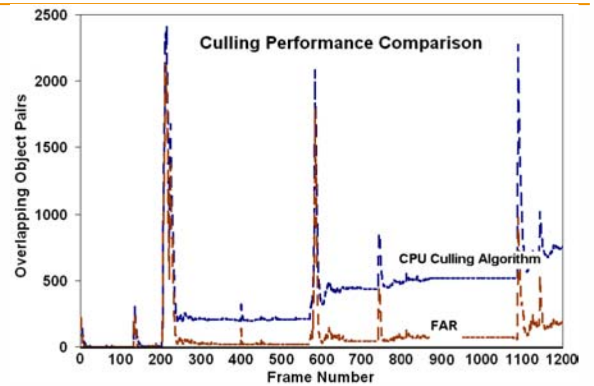
Collision time
vs.
screen resolution

NV30 GPU
Pentium IV 2 GHz CPU

Analysis: Pruning Efficiency

- Input complexity
- Relative object configurations
- Pruning efficiency in
  - Object-Level Culling
  - Subobject-Level Culling

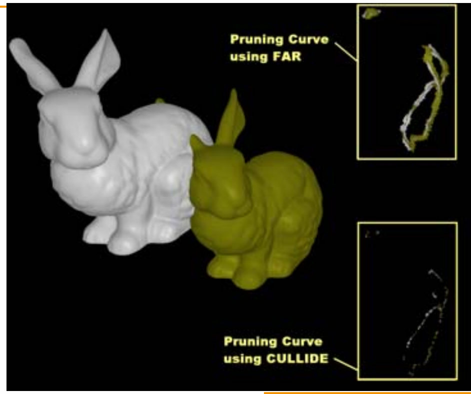
Comparison: FAR and I-COLLIDE



Analysis: Accuracy

- CULLIDE and S-CULLIDE: Image resolution
- FAR: IEEE 32-bit floating-point precision
- Comparison:
  - FAR vs. CULLIDE

Accuracy: FAR vs. CULLIDE



Outline

- Overview
- Interactive Collision Detection
- Conclusions and Future Work

Conclusions

- Designed efficient algorithms for solving
  - interactive collision detection,
  - shadow generation
- Applied them to complex 3-D environments
- Compared to prior state-of-the-art algorithms
  - Significant speedups in some cases

Advantages

- Generality
- Accuracy
  - IEEE 32-bit floating-point precision for collision computations
- Low Bandwidth
  - No readbacks

Advantages

- Significant Culling
- Practicality
  - Designed on commodity hardware
  - Assumes availability of occlusion queries

Limitations

- Precision
  - Shadow and self-collision algorithms are limited by image-precision
  - Accuracy can be improved
- Pair computation
  - Algorithms compute potential sets

Future Work

- Collision Detection
  - Pair computation
  - More applications – continuous collision detection, shadow volumes
  - Reliable self-collisions for general and specialized models
  - New programmability features

Future Work

- Shadow generation
  - Soft shadow generation

Future Work

- Visibility algorithms for
  - Line-of-sight
  - Database operations [Govindaraju et al. 2004]
  - Data mining [Govindaraju et al. 2005a]
  - 3-D sorting [Govindaraju et al. 2005b]
  - Order-statistics

Acknowledgements

- Student collaborators
  - Brandon Lloyd
  - Avneesh Sud
  - Stephane Redon (Post-Doctoral Researcher)
  - UNC Walkthrough, Gamma

Acknowledgements

- Army Research Office
- National Science Foundation
- Naval Research Laboratory
- Intel Corporation
- NVIDIA Corporation
  - Paul Keller and Stephen Ehmann for driver support

Thank You

- Questions or Comments?

naga@cs.unc.edu

http://gamma.cs.unc.edu/CULLIDE
http://gamma.cs.unc.edu/RCULLIDE
http://gamma.cs.unc.edu/QCULLIDE

Tutorial:

Real-Time Collision Detection for Dynamic Virtual Environments

Bounding Volume Hierarchies

Stefan Kimmerle   Johannes Mezger
WSI/GRIS
University of Tübingen

Outline

Introduction

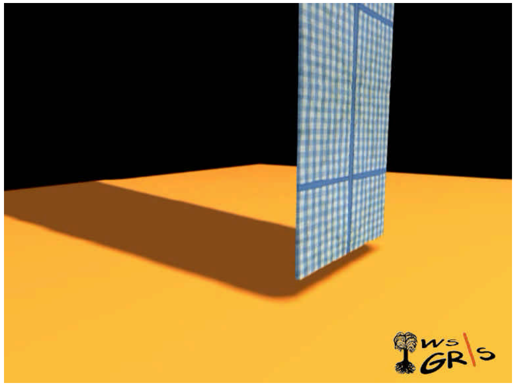
Problem of Collision Detection:

Object representations in simulation environments
do not consider impenetrability.

Collision Detection: Detection of interpenetrating objects.

The problem is encountered in
- computer-aided design and machining (CAD/CAM),
- robotics,
- automation, manufacturing,
- computer graphics,
- animation and computer simulated environments.

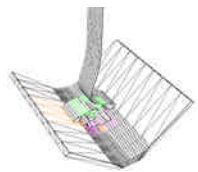
Introduction



Introduction

Definition of Bounding Volume Hierarchy (BVH):

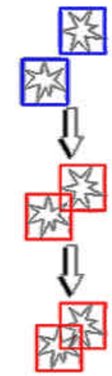
Each node of a tree is associated with a subset of primitives of the objects together with a bounding volume (BV) that encloses this subset with the smallest instance of some specified class of shape.



Introduction

Use these BVs as simplified surface representation for fast approximate collision detection test:

Examples of BVs:
- **Spheres**
- **Discrete oriented polytopes (k-DOPs)**
  Axis-aligned bounding boxes (AABB)
- **Object-oriented bounding boxes (OBB)**



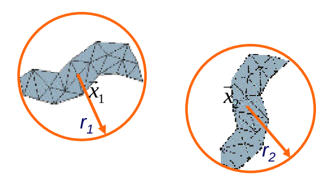
- Check bounding volumes to get the information whether bounded objects **could** interfere.
- Avoid checking all object primitives against each other.
- Assumption that collisions between objects are rare.

Outline

- Introduction
- Bounding Volume Types
- Hierarchy
  - Hierarchy Construction
  - Hierarchy Update
  - Hierarchy Traversal
- Comparison Rigid-Deformable Objects
- Examples and Conclusion

Bounding Volumes

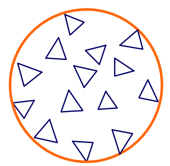
Spheres are represented by center $\vec{x}$ and radius *r*.

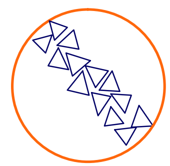


Two spheres do not overlap if $(\vec{x}_1-\vec{x}_2)\cdot(\vec{x}_1-\vec{x}_2) > (r_1+r_2)^2$

Bounding Volumes

Sphere as bounding volume:


good choice

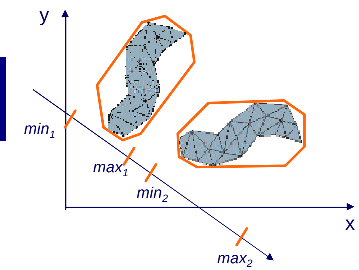

bad choice

Bounding Volumes

Discrete oriented polytopes (*k*-DOP) are a generalization of axis aligned bounding boxes (AABB) defined by *k* hyperplanes with normals in **discrete** directions ($n_k: \; n_{k,j} \in \{0, \pm 1\}$)

k-DOP is defined by *k*/2 pairs of *min*, *max* values in *k* directions.

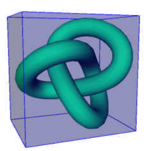
Two *k*-DOPs do not overlap, if the intervals in one direction do not overlap.



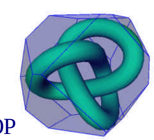
Bounding Volumes

Different ***k*-DOPs**:

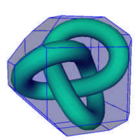
6-DOP (AABB)



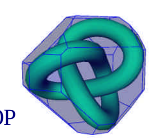
14-DOP



18-DOP

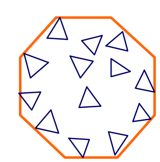


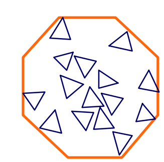
26-DOP



Bounding Volumes

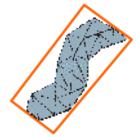
14-DOP as bounding volume:


optimal choice


also good choice

Bounding Volumes

Object oriented bounding boxes (OBB) can be represented by the principal axes of a set of vertices. These axes have **no discrete orientation**. They move together with the object.



The axes are given by the Eigenvectors of the covariance matrix:

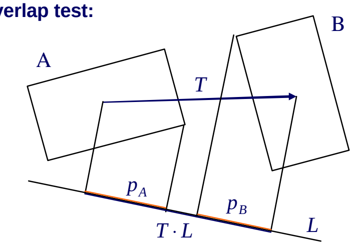
Centre of vertices $\vec{x}_i$: $\vec{m} = \frac{1}{n}\sum_{i=1}^{n}\vec{x}_i$

Covariance matrix: $C_{jk} = \frac{1}{n}\sum_{i=1}^{n}\bar{x}_{ij}\bar{x}_{ik}$ $\quad j,k = 1..3$

$\bar{x}_i = \vec{x}_i - \vec{m}$

Bounding Volumes

OBB overlap test:

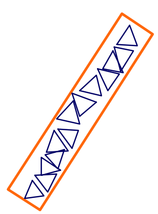


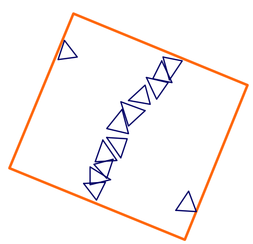
A and *B* do not overlap if: $\exists L : |T \cdot L| > p_A + p_B$

Problem: Find direction of *L*

Bounding Volumes

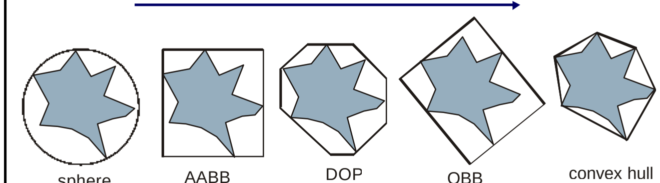
- Principal axes of an object are not always a good choice for the main axes of an OBB!
- Inhomogeneous vertex distribution can cause bad OBBs.





Bounding Volumes

Better approximation,
higher build and update costs



Smaller computational costs
for overlap test

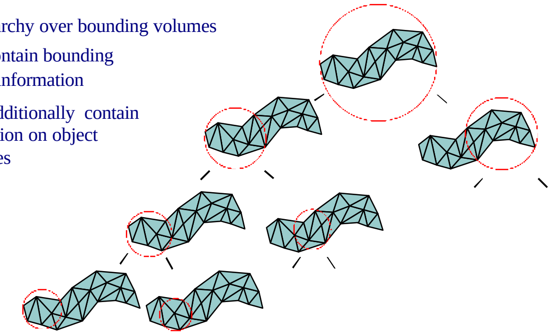
Outline

- Introduction
- Bounding Volume Types
- Hierarchy
  - Hierarchy Construction
  - Hierarchy Update
  - Hierarchy Traversal
- Comparison Rigid-Deformable Objects
- Examples and Conclusion

Bounding Volume Hierarchy

To further accelerate collision detection:

- use hierarchy over bounding volumes
- nodes contain bounding volume information
- leaves additionally contain information on object primitives



Hierarchy Construction

Parameters

- Bounding volume
- Type of tree (binary, 4-ary, k-d-tree, …)
- Bottom-up/top-down
- Heuristic to subdivide/group object primitives or bounding volumes
- How many primitives in each leaf of the BV tree

Goals

- Balanced tree
- Tight-fitting bounding volumes
- Minimal redundancy (primitives in more than one BV per level)

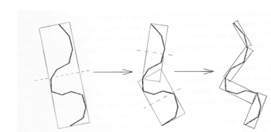
Hierarchy Construction

Bottom-Up

- Start with object-representing primitives
- Fit a bounding volume to given number of primitives
- Group primitives and bounding volumes recursively
- Stop in case of a single bounding volume at a hierarchy level

Top-Down

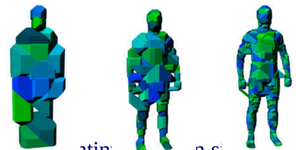
- Start with object
- Fit a bounding volume to the object
- Split object and bounding volume recursively according to heuristic
- Stop, if all bounding volumes in a level contain less than *n* primitives



Hierarchy Construction

Top-Down Node-split:

- Split *k*-DOP using heuristic:
  - Try to minimize volume of children (Zachmann VRST02).
  - Split along the longest side of the k-DOP (Mezger et al. WSCG03).

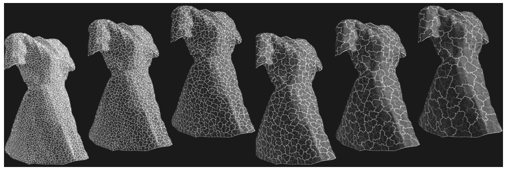


- The sp[illegible] contin[illegible] n single elements remain per leaf.

Hierarchy Construction

Bottom-Up Node-grouping:

- Group nodes using heuristic:
  - Try to get round-shaped patches by improving a shape factor for the area (Volino et al. CGF94).



- Group until all elements are grouped and the root node of the hierarchy is reached.

Hierarchy Update

Updating is necessary in each time step due to movement/deformation of simulated object.

Difference between rigid and deformable objects:

- For rigid objects: transformations can be applied to complete object.
- For deformable objects: all BVs need to be updates separately.
  - Update is possible top-down or bottom-up.
  - To avoid a complete update of all nodes in each step, different update strategies have been proposed.

Hierarchy Update

Some object transformations can be simply applied to all elements of the bounding-volume tree:

Spheres

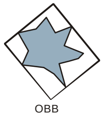
- Translation, rotation

Discrete Orientation Polytopes

- Translation, no rotation
  (discrete orientations of *k* hyperplanes for all objects)

Object-Oriented Bounding Boxes

- Translation, rotation
  (box orientations are not fixed)



Hierarchy Update

Larsson and Akenine-Möller (EG 2001):

- If many deep nodes are reached, bottom-up update is faster.
- For only some deep nodes reached, top-down update is faster.

-> Update top half of hierarchy bottom-up

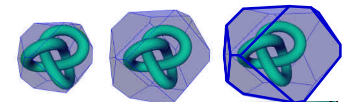
-> only if non-updated nodes are reached update them top-down.

- Reduction of unnecessarily updated nodes!
- Leaf information of vertices/faces has to be stored also in internal nodes -> higher memory requirements.

Hierarchy Update

Mezger et al. (WSCG 2003):

- Inflate bounding volumes by a certain distance depending on velocity.



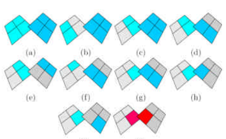
Update is only necessary if enclosed c ed farther than that distance.

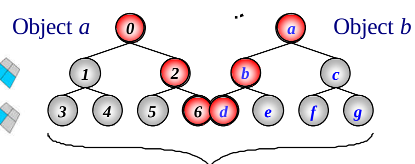
-> Fewer updates necessary.

-> More false positive collisions of BVs.

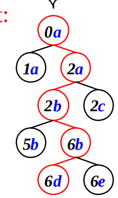
Hierarchy Traversal

Binary trees:





Collision test:

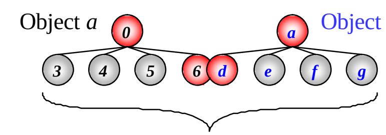


Minimize probability of intersection as fast as possible:

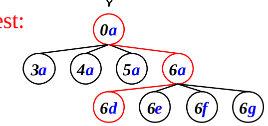
- Test node with smaller volume against the children of the node with larger volume.

Hierarchy Traversal

4-ary Trees:



Collision test:



Higher order trees:

- Fewer nodes
- Total update costs are lower
- Recursion depth during overlap tests is lower, therefore lower memory requirements on stack

Outline

- Introduction
- Bounding Volume Types
- Hierarchy
  - Hierarchy Construction
  - Hierarchy Update
  - Hierarchy Traversal
- Comparison Rigid-Deformable Objects
- Examples and Conclusion

Comparison – Collision Detection for Rigid and Deformable Objects

Rigid Objects:

- use OBBs as they are usually tighter fitting and can be updated by applying translations and rotations.
- update complete BVH by applying transformations
- usually small number of collisions occur

Deformable Object:

- use DOPs as update costs are lower than for OBBs
- update by refitting or rebuilding each BV separately (top-down, bottom-up)
- high number of collisions may occur
- Self-collisions need to be detected
- use higher oder trees (4-ary, 8-ary)

Outline

- Introduction
- Bounding Volume Types
- Hierarchy
  - Hierarchy Construction
  - Hierarchy Update
  - Hierarchy Traversal
- Comparison Rigid-Deformable Objects
- Examples and Conclusion

Example

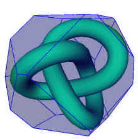
Interactive Cutting and Sewing

Conclusions

- BVHs are well-suited for animations or interactive applications, since updating can be done very efficiently.
- BVHs can be used to detect self-collisions of deformable objects while applying additional heuristics to accelerate this process.
- BVHs work with triangles or tetrahedrons which allow for a more sophisticated collision response compared to a pure vertex-based response.
- Optimal BVH and BV dependent on application (collision or proximity detection) and type of objects (rigid / deformable object)

Thank you …

Thank you!



Thanks to Matthias Teschner (University of Freiburg) and Johannes Mezger (University of Tübingen) for contributions to the slides!

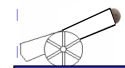
Stochastic Methods

Gabriel Zachmann
Universität Bonn

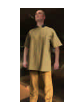
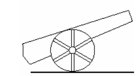
Motivation

- Absolute exactness not always necessary
- Real-time more important

→ Approximate collision detection



- Whenever only qualitative result matters, e.g.,
  Set of „plausible paths" for a cannonball.
- Games, virtual clothes prototyping, medical training, …

Overview

1. ADB-Trees [Klein & Zachmann, 2003]
2. Stochastic Closest Features Tracking
   [Raghupathi et al., 2004; Debunne & Guy, 2004]

ADB-Trees

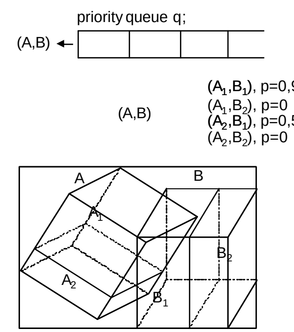
- ADB = "Average Distribution Trees"
- Average-case appraoch:
  - Estimate probability of intersection of 2 sets of polygons
- Applicable to almost any BV hierarchy
- Augment BVH by simple description of polygon distribution at inner nodes
- Probability-guided BVH traversal (p-queue)

Probability-Guided BVH Traversal

Traverse(A,B)

q.insert(A,B,1)
while q not empty
  A,B ← q.pop
  forall A_i, B_j
    p ← Pr[collision in A_i, B_j]
    if p ≥ pmin
      return "collision"
    if p ≥ 0
      q.insert(A_i, B_j, p)
 return "no collision"

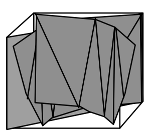
Probability-Guided BVH Traversal



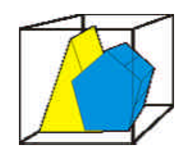
Traverse(A,B)

p-queue q
q.insert(A,B,1)
while q not empty
  A,B ← q.pop
  forall A_i, B_j
    p ← Pr[collision in A_i, B_j]
    if p ≥ p_{min}
      return "collision"
    if p ≥ 0
      q.insert(A_i, B_j, p)
 return "no collision"

Well-filled Cells and Collision Cells



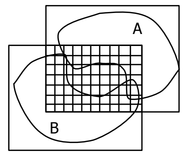
"well-filled" cell



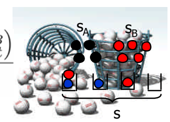
possible collision cell

Computing the Probability of Intersection

1. Partition A ∩ B by grid with s cells
2. Determine number of "well-filled" cells from BV A: s_A
3. Dito for B: s_B
4. Compute probability that x cells are well-filled from A and from B:

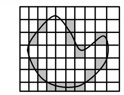


$$Pr[c(A \cap B) \geq x] = 1 - \sum_{t=0}^{x-1} \frac{\binom{s_B}{t}\binom{s-s_B}{s_A-t}}{\binom{s}{s_A}}$$

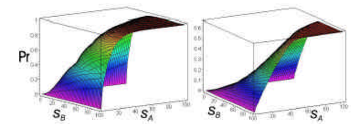


- Take curvature within cell into account:

$$\max_{x \leq \min\{s_A, s_B\}} \{Pr[c(A \cap B) \geq x] \cdot (1 - (1 - LB(A \cap B))^x)\}$$

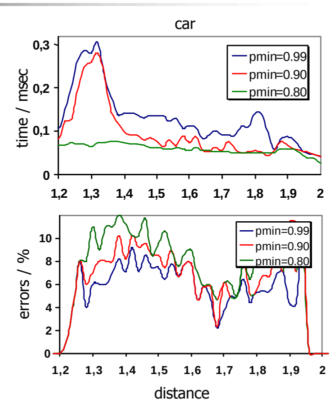


- Preprocessing
- Estimate parameters
- Lookup-tables for probability functions:



Result

- Time vs. error:

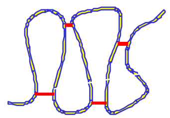


Stochastic Closest Features Tracking

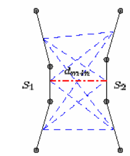
- Based on Lin-Canny (only for convex objects)
  - Steepest descent for single pair of features
  - Accelerated by generalized Voronoi diagram
  - Temporal coherence
- Extension to non-convex, deformable objects:
  - Non-convex → multiple pairs of (locally) closest features
  - Deformable → feature pairs come and go
  - Voronoi diagram not really feasible
- Idea
  - Stochastically create pairs of features
  - Converge them to locally closest features

Details

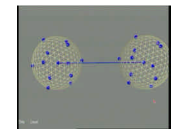
- Algorithm:
  Do animation step
  Add random pairs to list of "active feature pairs"
  For each feature-pair:
  Update features by local search
  Remove "unwanted" pairs
  If collision:
  apply response to local collision area



- Updating of feature pair:
  - No Voronoi diagram
  - Compute pairwise distance of all neighbor pairs

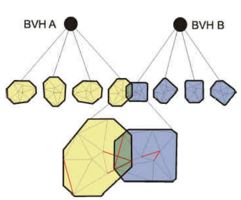


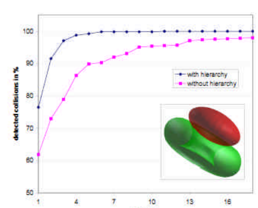
- Removal of feature pairs:
  - Distance too large (not likely closest feature)
  - Both features of two pairs too close (redundant)
- Creation of feature pairs:
  - Importance-driven (e.g., velocity-based)
  - Supported naturally by multires model



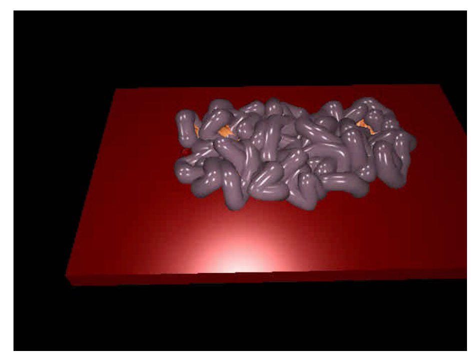
Acceleration by BV Hierarchy

- Use incomplete BVH to find "interesting" regions
- Stochastically sample those regions

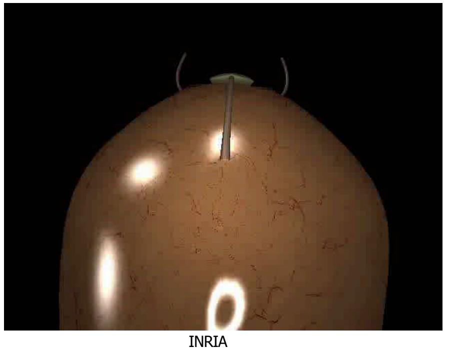




Example Application



INRIA



INRIA

Conclusions

- Stochastic methods are not always error-free
- Good for plausible & fast simulations
- Interesting alternative to BVHs in specific cases
- Naturally yield time-critical collision detection
- Future work:
  - Continuous stochastic methods
  - Precise error bounds

Collision Detection for Deformable Objects
-
Distance Fields

Arnulph Fuhrmann

Fraunhofer Institute for Computer Graphics

Darmstadt, Germany

Outline

- Introduction
- Distance Field Generation
- Collision Detection using Distance Fields
- Conclusion

Introduction

- Physically based modeling
  - Cloth, hair, etc.
- Problem
  - Many contact points
- During Simulation
  - Detect Collision
  - Compute Collision Response
    - Proximity or penetration depth
    - Surface normal

Distance Field Definition

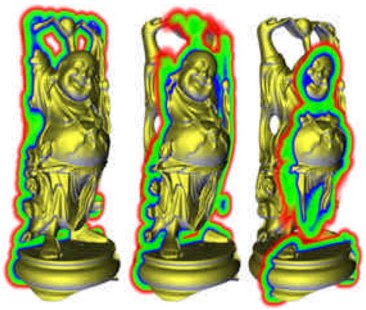
- Scalar function

$$D : \mathbb{R}^3 \rightarrow \mathbb{R}$$

- $dist(\mathbf{p})$ = distance to closest point on surface
- $sign(\mathbf{p})$ = negative if inside object

$$D(\mathbf{p}) = sign(\mathbf{p}) \cdot dist(\mathbf{p})$$

Example – Distance Field 2D-Slices

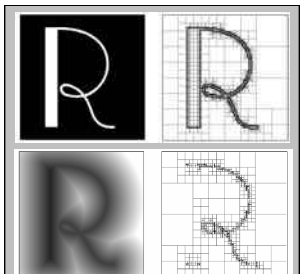


Outline

- Introduction
- **Distance Field Generation**
- Collision Detection using Distance Fields
- Conclusion

Distance Field Data Structures

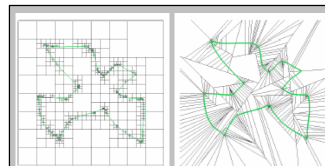
- Uniform 3D grid
  - Queries take *O(1)* time
  - Curved surfaces can be represented quite well
  - C^0 continuous
- Adaptively sampled distance fields (ADFs)
  - [Frisken et al. '00]
  - C^{-1} between different levels
    - can be resolved



Frisken et al. '00

Distance Field Data Structures

- BSP-tree
  - [Wu and Kobbelt '03]
  - Piecewise linear approximation
  - Generation computationally expensive
  - Discontinuities between cells

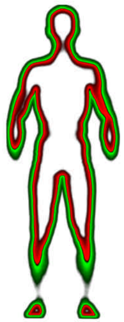


ADF 895 cells   BSP-tree 254 cells

[Wu and Kobbelt '03]

Computation of Distance Fields

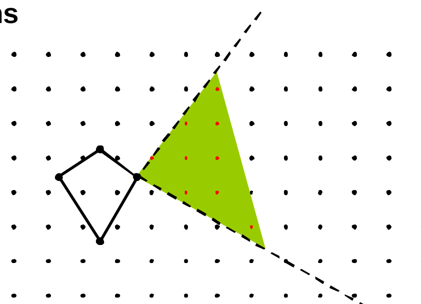
- Object representation
  - triangular mesh
- Problem
  - Computing distances for all grid points
  - Naïve computation too costly
- Collision detection
  - only a small band needed



Computation of Distance Fields

- Propagation methods
  - Fast Marching methods [Sethian '96]
  - Distance Transforms [Jones and Satherley '01]
- Rasterizing of distance functions
  - Full distance field
  - [Sud et al. '04], [Hoff et al. '99]
- Bounded Voronoi Regions
  - [Sigg et al. '03], [Breen et al. '01]
  - bounding polyhedron around Voronoi regions of edges, faces and vertices

Scan Conversion of Bounded Voronoi Regions

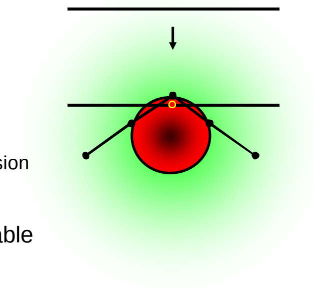


Outline

- Introduction
- Distance Field Generation
- **Collision Detection using Distance Fields**
- Conclusion

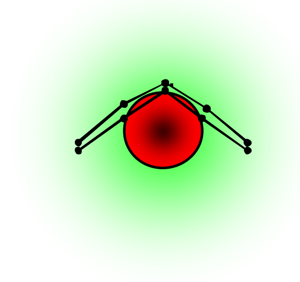
Collision Detection

- Scenario
  - Deformable object A
  - Static object B
- Collision Detection
  - Sample object A
  - Test sample points for collision with B
- If both objects are deformable
  - Swap and repeat



Collision Detection

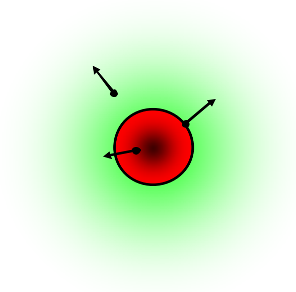
- Problem
  - Edges intersect object
- Solution
  - Preserve ε distance at vertices



Queries needed for collision detection

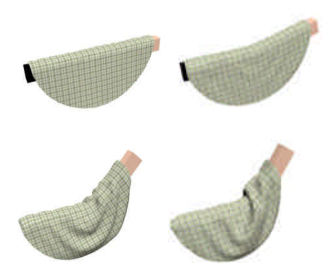
(On a uniform 3D grid)

- Distance
  - Tri-linear interpolation
- Normal
  - Direction given by the gradient



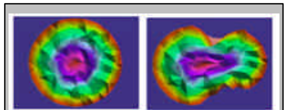
What about deforming collision objects?

- Multiple distance fields
- Linked rigid objects
  - One distance field per object
- Not possible yet
  - Soft objects like a bending human arm



Other approaches for deforming objects

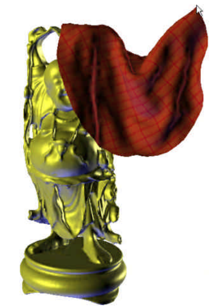
- [Bridson et al. '03]
  - Clothing and animated characters
  - Pre-computed ADFs for the body parts
  - Can be used for several cloth simulations
- [Fisher and Lin '01]
  - Deforming geometries
  - Collision detection is done hierarchically
  - Partial DF updates only
  - Internal distance fields for collision response



[Fisher and Lin '01

Demo Video

- Captured directly from screen
- Implemented in Java 1.4.1 and Java3D 1.2
- Tests made on a Intel Xeon Processor at 2.0 GHz
- Buddha model consist of 390.000 triangles!



Distance Fields for Rapid Collision Detection in Physically Based Modeling

Fraunhofer Institut Graphische Datenverarbeitung

Outline

- Introduction
- Distance Field Generation
- Collision Detection using Distance Fields
- **Conclusion**

Summary

- Distance Fields Generation
  - Pre-Processing step
  - Duration: Some seconds
- Collision Detection using Distance Fields
  - Most useful for deformable against rigid objects
  - Efficient computation of
    - Penetration depth / proximity
    - Gradient (Normal)
  - Easy to implement
  - Robust algorithm