Collision Detection
2004

Datorgrafik

Stefan Seipel (stefan.seipel@hig.se)
Purposes of collision detection

Computer Animation and Offline-Rendering
   Modeling interaction of light, matter and objects
      (light tracing, shadow calculation)

Visual Simulation (engineering applications)
   Modeling interaction between objects in the scene
      (hit testing, event creation, physical behavior modeling)

Interactive Graphical Systems (VR, games)
   Modeling interaction between objects in the scene
      (hit testing, event creation, physical behavior modeling)
   Interacting between user and objects
Requirements to collision detection

Computer Animation and Offline-Rendering

Highest possible accuracy
Many parameters desired (hit, position, angles, texture coordinates)
Timing aspect not so critical

Visual Simulation (engineering applications)

Highest possible accuracy
Many (physical) parameters desired (hit position, angles, velocity, mass-momentum)
Timing aspects important

Interactive Graphical Systems (VR, games)

Real-time aspect most important
Accuracy often not predominant
Little parameter density
Physical behavior often approximated with simple functions
Collision Detection Approaches

Analytical Per-Primitive Methods:
• Every entity in a scene represented as parametric object (e.g. polygon, parametric function)
• Exact tests are performed based on linear algebra methods
• Accurate solutions, time consuming
• Final stage tests on pre-selected primitive subset

Approximation Methods:
• Discrete logic or algebraic solutions, low specificity, reliable for negative result
• Bounding volumes, simplified representations
• First stage testing to sort out non-intersecting objects
• Far field testing

Hardware Supported Methods:
• Used hardware implemented features of transformation pipeline or rasterizer unit
• Require multi-pass rendering i.e. frame rate might drop
• May suffer from inaccuracy (whenever based on discrete buffers)
Analytical Hit Tests

• Point hits plane \( \Leftrightarrow \) Distance point - plane & sphere hits plane (state testing)
• Intersection ray and plane (vector based notion)
• Intersection ray and plane (parametric normal form)
• Sphere intersects with sphere
• Sphere hits plane (position prediction)
• Sphere hits sphere (position prediction)
• Intersection ray and triangle (based on vector based notion)
• Intersection ray and triangle (based on parametric solution)
• Triangle intersects Triangle
Point hits plane - distance point-plane - sphere hits plane

A) Assumption:
Plane with normal vector N and point P on the plane known.
Test point T given.

Calculation of shortest distance between T and plane:
There is a vector PT from P to T. The length of the projection of PT upon N can be calculated using the scalar product:

\[ PT = (tx-x, ty-y, tz-z), \quad N = (nx, ny, nz) \]

\[ dt = (tx-x) * nx + (ty-y) * ny + (tz-z) * nz \]

B) Assumption:
Plane is given in normal form:
Plane is given in normal form:
\[ Ax + By + Cz + D = 0 \]
(where A,B,C normal vector components, and shortest distance of plane to origin is known \( D = -d \))

Then insert T in plane equation and check result: if \( dt = 0 \), then T is on plane

\[ dt = Atx * Bty + Ctz + D = 0 ? \]
Intersection ray and plane (vector based notion)

Test with reliable result both for positive and negative result.

Assumption: Given three points \( v_1, v_2, \) and \( v_3 \) on the plane.
Given ray position \( P \) and direction vector \( d \).

Span-vectors:
\[
\begin{align*}
  s_1 &= v_2 - v_1; \\
  s_2 &= v_3 - v_1;
\end{align*}
\]

Then:
\[
P + \tau \cdot d = v_1 + \lambda \cdot s_1 + \mu \cdot s_2
\]

Cramer, Gauß

one solution -> exactly one hit point
no solution -> ray parallel to plane
many solutions -> ray within plane
Intersection ray and plane (parametric normal form)

The plane:
\[ N \cdot P = d \quad \land |N| = 1 \]
\[ \begin{align*}
Ax + By + Dz + D &= 0 \\
N &= \begin{pmatrix} A \\ B \\ C \end{pmatrix} \quad \land D = -d
\end{align*} \]

The ray:
\[ P = RS + \tau \cdot RD \]

If for a point \( P \) both on plane and ray:
\[ N \cdot (RS + \tau \cdot RD) = d \iff N \cdot RS + \tau (N \cdot RD) = d \]

Solving for \( \tau \):
\[ \tau = \frac{(d - N \cdot RS)}{N \cdot RD} = \frac{(N \cdot P - N \cdot RS)}{N \cdot RD} = \frac{N \cdot (P - RS)}{N \cdot RD} \]

Result: Parameter \( \tau \) i.e. a point in 3D space (no more parameters)!
Sphere intersect with sphere

Collision test:
Two spheres intersect, if the distance of their midpoints is equal or less than the sum of their radius.

Assumption: $C^A$ and $C^B$ center coordinates of two spheres in $R^3$, and radius $R^A$ and $R^B$ scalar values.

Then:

$$d \leq R^A + R^B$$

$$d \iff |C^B - C^A|$$

$$\sqrt{(c_x^B - c_x^A)^2 + (c_y^B - c_y^A)^2 + (c_z^B - c_z^A)^2} \leq R^A + R^B$$
Sphere hits plane (position prediction)

Known:
- Sphere radius \( r \)
- Sphere center position \( CS \)
- Sphere motion direction \( D \)
- Point on plane and plane normal

Approach:
If the sphere will hit the plane
this will happen at point \( PS \)
which is the point on the sphere
on the shortest way towards the plane.

Therefore:
\[
PS = CS - rN
\]
and
\[
Hit = (CS - rN) + \tau D
\]

Solution for \( \tau \) same as in
ray-plane intersection testing.
(see above)
Sphere hits sphere (position prediction)

For pre-defined trajectories possible.

Case of linear motion:

Theoretically the sweep volumes of the spheres intersect wherever shortest distance between central lines of motion is less than the sum of the radiuses.

There is an analytical solution
(see paper by Bard and Himel 2001)

However, what about velocity aspects?

Reverse calculation based on distance to-hit, in order to calculate time window.

What about non constant velocities?
-> iterative solution
Case of non-linear motion track:

Iterative solution:

For a limited time interval into the future step through predictable positions of object 1 and object 2.

Use a time step resolution, which is adapted to the max. velocity under this time period. Otherwise possible hits might be left out.

Time step resolution can be calculated so that eg. the maximum allowable $dx/dt$ is less than 0.5 of the smallest radius of both objects.
Intersect ray and triangle (based on vector based notion)

Test with reliable result both for positive and negative result.

Triangle: \( v_1, v_2, \) and \( v_3 \)

Span-vectors: \( s_1 = v_2 - v_1 \)
\( s_2 = v_3 - v_1 \)

Ray: \( P, \) and direction vector \( d \)

Then:

\[
P + \tau \cdot d = v_1 + \lambda \cdot s_1 + \mu \cdot s_2
\]

\[
\text{LES}
\]

\[
\text{Cramer, Gauß}
\]
1 Solution?

\[
\left( \tau, \lambda, \mu \right) \Rightarrow 0 \leq (\lambda + \mu) \leq 1
\]

Method allows for texture coordinate generation!

©2003 Stefan Seipel
3D point inside triangle?

Parametric solution only yields a point on the plane of v1..v3.

Is the point inside the triangle?

If the sum of the angles between

\[ a \text{ and } c \]
\[ a \text{ and } b \]
\[ b \text{ and } c \]

is 360 degrees, then hit is within the triangle.

\[ a = v1 - hit \]
\[ b = v2 - hit \]
\[ c = v3 - hit \]

\( \hat{a}, \hat{b}, \hat{c} \) the normalized vectors

\[ \alpha = \arccos(\hat{a} \cdot \hat{c}) + \arccos(\hat{a} \cdot \hat{b}) + \arccos(\hat{b} \cdot \hat{c}) \equiv 360 + \varepsilon \]

Where \( \varepsilon \) tolerance value

Method allows NOT for texture coordinate generation!
Triangle intersects triangle

Test with reliable result both for positive and negative result.

Two triangles $T_1$, $T_2$ intersect, if at least one edge of $T_1$ intersects with $T_2$ or if one edge of $T_2$ intersects with $T_1$

Perform 6 times a ray-triangle test and check LES result for:

$$0 \leq (\lambda + \mu) \leq 1 \land 0 \leq (\tau) \leq 1$$

Any edge of $T_2$ pierces $T_1$?

1. Test: $T = v_1, v_2, v_3$
   $R = v_4, (v_6-v_4)$

2. Test: $T = v_1, v_2, v_3$
   $R = v_4, (v_5-v_4)$

3. Test: $T = v_1, v_2, v_3$
   $R = v_5, (v_6-v_5)$

Any edge of $T_1$ pierces $T_2$?

4. Test: $T = v_4, v_5, v_6$
   $R = v_1, (v_3-v_1)$

5. Test: $T = v_4, v_5, v_6$
   $R = v_1, (v_2-v_1)$

6. Test: $T = v_4, v_5, v_6$
   $R = v_2, (v_3-v_2)$
Complexity considerations (e.g. triangle intersection testing)

Objects, and each object has np_i triangles. Collision detection on a per-polygon basis.

Worst-Case scenario for triangle intersection tests:

Total number of triangle-triangle tests:

\[ tnt = \sum_{i=1}^{n-1} \left( np_i \sum_{k=i+1}^{n} np_k \right) \]

**Example:** Four objects containing 300, 400, 200, and 1000 triangles.

\[ tnt = 300*(400+200+1000) + 400*(200+1000) + 200*1000 = \]
\[ 300*1600 + 400*1200 + 200*1000 = \]
\[ 480000 + 480000 + 200000 = 1160000 \]

=> 6.6 Mio LES
Approximation approaches

Bounding box collision test
Bounding sphere collision tests
Hierarchical bounding volume tests (e.g. spheres)
Multiple level-of-detail collision tests
Spatial occupancy matrix and space voxelization
Hot spot collision testing
Collision test in image based rendering
Bounding Box Intersection Testing: Initialization

Bounding Box Calculation:

For any two objects A, B in the world coordinate system

-> search maximum/minimum x, y, and z coordinates.

-> Bounding box A given by 8 vertices:

\[ V_1 = (A_{mix}, A_{miy}, A_{miz}) \]
\[ V_2 = (A_{mix}, A_{miy}, A_{maz}) \]
\[ V_3 = (A_{mix}, A_{may}, A_{miz}) \]
\[ V_4 = (A_{mix}, A_{may}, A_{maz}) \]
\[ V_5 = (A_{max}, A_{miy}, A_{miz}) \]
\[ V_6 = (A_{max}, A_{miy}, A_{maz}) \]
\[ V_7 = (A_{max}, A_{may}, A_{miz}) \]
\[ V_8 = (A_{max}, A_{may}, A_{maz}) \]

Bounding box B given by 8 vertices:

\[ V_1 = (B_{mix}, B_{miy}, B_{miz}) \]
\[ V_2 = (B_{mix}, B_{miy}, B_{maz}) \]
\[ V_3 = (B_{mix}, B_{may}, B_{miz}) \]
\[ V_4 = (B_{mix}, B_{may}, B_{maz}) \]
\[ V_5 = (B_{max}, B_{miy}, B_{miz}) \]
\[ V_6 = (B_{max}, B_{miy}, B_{maz}) \]
\[ V_7 = (B_{max}, B_{may}, B_{miz}) \]
\[ V_8 = (B_{max}, B_{may}, B_{maz}) \]
Bounding Box Intersection Test

Bounding Box Test:

For any two bounding boxes A, B: Test all vertices in B for inside check A:

\[
\begin{align*}
(A_{\text{mix}} < B_{\text{mix}} < A_{\text{max}}) \land (A_{\text{miy}} < B_{\text{miy}} < A_{\text{may}}) \land (A_{\text{miz}} < B_{\text{miz}} < A_{\text{maz}}) \\
\text{OR} \\
(A_{\text{mix}} < B_{\text{mix}} < A_{\text{max}}) \land (A_{\text{myy}} < B_{\text{miy}} < A_{\text{may}}) \land (A_{\text{miz}} < B_{\text{maz}} < A_{\text{maz}}) \\
\text{OR} \\
(A_{\text{mix}} < B_{\text{mix}} < A_{\text{max}}) \land (A_{\text{miy}} < B_{\text{may}} < A_{\text{may}}) \land (A_{\text{miz}} < B_{\text{miz}} < A_{\text{maz}}) \\
\text{OR} \\
(A_{\text{mix}} < B_{\text{max}} < A_{\text{max}}) \land (A_{\text{miy}} < B_{\text{miy}} < A_{\text{may}}) \land (A_{\text{miz}} < B_{\text{miz}} < A_{\text{maz}}) \\
\text{OR} \\
(A_{\text{mix}} < B_{\text{max}} < A_{\text{max}}) \land (A_{\text{myy}} < B_{\text{miy}} < A_{\text{may}}) \land (A_{\text{miz}} < B_{\text{maz}} < A_{\text{maz}}) \\
\text{OR} \\
(A_{\text{mix}} < B_{\text{max}} < A_{\text{max}}) \land (A_{\text{myy}} < B_{\text{may}} < A_{\text{may}}) \land (A_{\text{miz}} < B_{\text{maz}} < A_{\text{maz}})
\end{align*}
\]

Worst Case:

- 48 float point compares
- 23 logical operations
Bounding Box Intersection Testing: Comparison

**Advantages:**
- Simple arithmetic to test for collision
- Only bounding box corners need to be transformed on object motion
- Very fast test
- Bounding volume approximates elongated, flat objects quite well
- Negative test result is 100% reliable

**Disadvantages:**
- Bounding volume is *not invariant* with regard to rotation
- Object approximation can become quite inaccurate
- Positive test result is not specific
Bounding Sphere Intersection

Bounding Sphere Calculation:

1. For an object find the center of gravity \( C \) by calculating the average \( x, \) \( y, \) and \( z \) components of all vertices.

\[
C = \left( \frac{c_x}{c_y} \right) = \frac{1}{n} \sum_{i=1}^{n} V^i = \frac{1}{n} \left( \sum_{i=1}^{n} v^i_x \right) \left( \sum_{i=1}^{n} v^i_y \right) \left( \sum_{i=1}^{n} v^i_z \right)
\]

2. For all vertices in the object calculate the distance to \( C \) and keep the maximum distance value in mind. The maximum distance will become the bounding spheres radius.

\[
R = MAX(|V^i - C|) = MAX\left( \sqrt{(v^i_x - c_x)^2 + (v^i_y - c_y)^2 + (v^i_z - c_z)^2} \right)
\]
Collision detection:
Test any two bounding spheres A and B of two objects for intersection. Two spheres intersect, if the distance of their midpoints is equal or less than the sum of their radius:

\[ d \leq R^A + R^B \]

\[ d \Leftrightarrow \left| C^B - C^A \right| \]

\[
\sqrt{(c^B_x - c^A_x)^2 + (c^B_y - c^A_y)^2 + (c^B_z - c^A_z)^2} \leq R^A + R^B
\]
Bounding Sphere Intersection

Advantages:
• Test is very fast
  (requires 3 adds, 3 subs, 3 squares, 1 root)
• Bounding volume is invariant with regard
  to rotation -> can be calculated once at the
  beginning of geometry setup.
• Bounding volume approximates solid, symmetric objects quite well
• Negative test result is 100% reliable

Disadvantages:
• Approximation of linear or planar shaped objects is quite poor
• Test has low specificity
Hierarchical bounding volume tests

Approach:
- Use multiple (hierarchically) arranged bounding spheres to better approximate rigid objects.
- Use multiple hierarchically arranged bounding sphere system to approximate articulated objects. (e.g. robot arm systems….)

Tricky issue:
- Automatic generation of bounding sphere hierarchy
- Synchronization of model articulation and bounding sphere hierarchy update
Multiple level of detail (LOD) collision testing

Basic idea:
• Keep multiple geometric representations for a single object to be used in a scene
• Use models with high polygon count for close field of view rendering
• Use models with low polygon count for collision detection and far field of view rendering

Specific aspects:
• How much model degradation is allowable?
  • from a visual point of view
  • with regard to collision accuracy

• How much does it pay off?
Multiple LOD collision testing: Allowable degradation

Detail Level 1: 7446 triangles

Detail Level 2: 3724 triangles

Detail Level 3: 746 triangles

Detail Level 4: 500 triangles
Multiple LOD collision testing: Performance aspects

Number of triangle tests to perform (brute force)

<table>
<thead>
<tr>
<th>Objects</th>
<th>Test pairs</th>
<th>LOD1: 7446</th>
<th>LOD2: 3724</th>
<th>LOD3: 746</th>
<th>LOD4: 500</th>
<th>LOD5: 300</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>1</td>
<td>55442916</td>
<td>13868176</td>
<td>556516</td>
<td>250000</td>
<td>90000</td>
</tr>
<tr>
<td>3</td>
<td>3</td>
<td>166328748</td>
<td>41604528</td>
<td>1669548</td>
<td>750000</td>
<td>270000</td>
</tr>
<tr>
<td>4</td>
<td>6</td>
<td>332657496</td>
<td>83209056</td>
<td>339096</td>
<td>1500000</td>
<td>540000</td>
</tr>
<tr>
<td>5</td>
<td>10</td>
<td>554429160</td>
<td>138681760</td>
<td>5565160</td>
<td>2500000</td>
<td>900000</td>
</tr>
<tr>
<td>6</td>
<td>16</td>
<td>831643740</td>
<td>208022640</td>
<td>8347740</td>
<td>3750000</td>
<td>1350000</td>
</tr>
</tbody>
</table>

Time in sec. to perform brute force testing (0,1 microseconds/test optimistically assumed)

<table>
<thead>
<tr>
<th>Objects</th>
<th>Test pairs</th>
<th>LOD1: 7446</th>
<th>LOD2: 3724</th>
<th>LOD3: 746</th>
<th>LOD4: 500</th>
<th>LOD5: 300</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>1</td>
<td>5,5442916</td>
<td>1,3868176</td>
<td>0,0556516</td>
<td>0,025</td>
<td>0,009</td>
</tr>
<tr>
<td>3</td>
<td>3</td>
<td>16,6328748</td>
<td>4,1604528</td>
<td>0,1669548</td>
<td>0,075</td>
<td>0,027</td>
</tr>
<tr>
<td>4</td>
<td>6</td>
<td>33,2657496</td>
<td>8,3209056</td>
<td>0,339096</td>
<td>0,15</td>
<td>0,054</td>
</tr>
<tr>
<td>5</td>
<td>10</td>
<td>55,442916</td>
<td>13,868176</td>
<td>0,556516</td>
<td>0,25</td>
<td>0,09</td>
</tr>
<tr>
<td>6</td>
<td>15</td>
<td>83,164374</td>
<td>20,802264</td>
<td>0,834774</td>
<td>0,375</td>
<td>0,135</td>
</tr>
</tbody>
</table>

Assumption: All objects have identical number of polygons.
Multiple LOD collision testing: Tricky aspects

Who performs mesh minimization?

Which mesh minimization criteria are applied?
(shape preserving ???)

Full enclosure envelope? False-Negative Results!

At what LOD does shape degrade too much?

Is model suitable for multiple LOD representation?

Detail Level 4: 500 triangles

Detail Level 5: 300 triangles
Morphology changes significantly !!!
Spatial occupancy matrix and space voxelization

**Example scenario:** Many objects moving within a wide spatial region
(submarine scenario, terrain simulation, flight surveillance)

**Goal:** Avoiding the brute-force testing method (pairing all objects in scene)

**Problem:** To identify which object pairs are likely to collide?

**Approach:**
In order to make selection of object pairs which are likely to intersect, simulation space is partitioned in regularly arranged clusters.

At each simulation step, global positions for all objects are mapped upon cluster address.

Addressed cluster (e.g. voxel) keeps reference to objects within this cluster.

Cluster size and number depends on:
- maximum object size
- total extend of the scene
- memory available
Hot-spot collision testing

Tavla
Collision test in image based rendering

Objects are drawn as alpha blended textures
E.g. trees, bushes…

Analytical tests yields positive result

However, hit is determined by texture fragment rendered at hit position

Calculation of texture coordinates required

Calculation of texel address

Lookup of texel value texture memory

Hit-coordinates in vector notation required
Hardware supported approaches

Pre-selection of objects in viewing volume
Picking using Selection Buffer (e.g. using OpenGL)
Picking using the BackBuffer (e.g. using OpenGL), Two Pass Rendering
Picking (analytical approach)
Pre-selection of visible objects using selection-buffer

Purpose:

• Reduce the number of tested objects (for analytical picking)
• Reduce the number of object-pairs (object collision testing)

Functional principle in OpenGL:

• Selection buffer, to carry names of primitives/objects drawn by OpenGL
• Name stack to push names
• OpenGL allows to load names onto the name stack
• Whenever under drawing a primitive passes the viewing frustum test
  the current name on the name stack is added to the selection buffer
  (if that is different from the last name in the selection buffer)
Pre-selection of visible objects using selection-buffer

Steps involved:

\[
\text{Gluint selection_buffer[BUFSIZE];} \\
\text{Glint hits;} \\
\text{glSelectBuffer(selection_buffer);} \quad \text{// pass a buffer to OpenGL to carry selection entries} \\
\text{glInitNames();} \quad \text{// initialize name stack} \\
\text{glPushName();} \quad \text{// pushing entry on name stack} \\
\text{SetUpViewingModellingMatrix} \\
\text{glRenderMode(GL_SELECT)} \\
\text{glLoadName(1)} \\
\text{DrawTriangle(1)} \\
\text{glLoadName(2)} \\
\text{DrawTriangle(2)} \\
\text{glLoadName(3)} \\
\text{DrawTriangle(3)} \\
\text{hits = glRenderMode(GL_RENDER)}
\]
Pre-selection of visible objects using selection-buffer

Selection record structure

1. Number n of names on stack, when hit occurred
2. Minimum window-z coordinate
3. Maximum window-z coordinate
4. Lists of n names of (content of name stack)

Our example:

hits = 2

```
selection_buffer[0] = 1
selection_buffer[1] = minz_1
selection_buffer[2] = maxz_1
selection_buffer[3] = 1

selection_buffer[4] = 1
selection_buffer[1] = minz_3
selection_buffer[3] = 3
```
Picking objects using the selection-buffer

Purpose: Identifying object beyond current cursor position

Special case of selection:

For a given cursor position on screen, a narrow viewing volume can be set up.

```c
glMatrixMode(GL_PROJECTION);
glPushMatrix();
glLoadIdentity();
gluPickMatrix(...); // prior to the rest of
    // the projection setup !!!
glPerspective(...), or whatever
    continue as above
```

see also

```c
glGetIntegerv(GL_VIEWPORT, Glint *viewport);
```

```c
gluPickMatrix(
    Gldouble x,
    Gldouble y,
    Gldouble width,
    Gldouble height,
    Glint viewport[4]
);
```
Picking objects using the back-buffer

Purpose: Identifying object beyond current cursor position

- Render full featured objects to back-buffer (i.e. lightning, material etc. activated)
- Swap buffers (visual result evident in front-buffer)
- Disable shading, illumination
- Encode object(s) identifier into color code and set rawing color
- Render object(s) into back buffer
- Read pixels from back buffer at cursor position (glReadPixles)
- Decode color code into object identifier

Features:
- Two-pass rendering approach, reduces effective frame rate
- Somewhat complicated implementation approach
- Utilized HW supported rasterizing and geometry transformation
- Exploits hardware FoV culling
Hardware Approaches: Discussion

- Very special pointing metaphor
- Ray: eye-cursor-target
- Rotational ray-control in screen space
- Hit-tests in screen domain
- No hits outside viewing volume
- No hits for culled polygons
- Easy to control with a 2D mouse
- Unusable for full scale or immersive VR

Alternative Solution:
- Picking ray modeled as part of the 3D scene
- Collision testing using analytical ray-object testing
Discrete techniques using regular grids

Event maps / semantic maps
Digital elevation maps
Distance maps
Semantic maps / event maps

Efficient Collision Detection / Model Representation

9x9 Discrete Matrix Space

Real World Coordinate Space

©2003 Stefan Seipel
Digital elevation maps (DEM) 

Example application: Terrain simulation scenarios

Assumption: Terrain is primarily given as a regular two-dimensional grid of height values $E[n,m]

Simulation consequences: DEM can not be rendered immediately within the 3D simulation. Therefore, a polygonal mesh must be re-created from the 2.5-dimensional elevation map.

=> tons of polygons.

Approach: Find a suitable transfer function, which maps 3D simulation coordinates into 2.5D DEM coordinates. I.e. create a discrete row and column index pair $[i,k]$ for a given $(x,y,z)$ coordinate in simulation coordinates space, where $xz$-plane usually coplanar with DEM.

Collision test:
1. Look-up elevation value at $e = E[i,k]$ and compare $y$ for interval $[e-\epsilon..e+\epsilon]$, where $\epsilon$ is a predefined fuzz value.

2. Look-up polygons in the terrain model which correspond to this matrix column/row and perform additional tests.
Digital elevation maps (DEM s) continued

The role of \( \varepsilon \): 

Mapping e.g.:

\[
\begin{align*}
  i &= \text{round}(x/k-\text{offs}x) \\
  k &= \text{round}(z/k-\text{offs}z)
\end{align*}
\]

Mapping from real world coordinates to matrix indexes causes aliasing

Value \( E[i,k] \) has indicates elevation in the neighborhood of the actual point \( P = (x,y,z) \)

\( \varepsilon \) defines vertical tolerance area to compensate for spatial aliasing

\( \varepsilon \) depending on largest possible quantisation error in \( xz \) - plane i.e. 0.5 grid cell size

\( \varepsilon \) depending on largest slope across cell

\( \varepsilon \) controls the sensitivity of the hit test
Distance maps

Application example: Occlusal simulation
Distance maps

Application scenario:
- Two highly complex 3D (>100,000 polygons) objects are moving towards each other with a roughly known direction.
- Collision detection must be performed with highest possible accuracy.
- Bounding volume techniques and LOD approaches not applicable.
- Quite common situation in e.g. medical simulations.

Problem:
- Extremely high polygon count requires other strategies than polygon intersection testing.

Approach:
- Have or create 2.5 dimensional representation for each object
- Distance map with high resolution where elevation coincides with the main direction of motion
- Implement a rasterizer engine which transforms one elevation map into the other one
- Per-pixel comparison of elevation values and collision detection
Distance maps (example)

Anatomic model of the teeth

About 300,000 polygons per jaw

Lower jaw motion is known

Definition of a projection plane which is perpendicular to dominant motion direction

Projection of jaw models towards defined plane and readout of distance values for any pixel covered by jaw fragment

Important issue: Keep spatial relation between jaws and reconstruction plane
Distance maps (example)

Can e.g. be created by rendering orthogonal with non-perspective viewing volume and by reading pixels from the z-buffer.
Distance maps

Collision detection:

• For any spatial relation between the two 3D objects, retrieve the mapping function to transfer pixels from one distance map into the other distance map.

• Rasterize every pixel of one depth map into the other map.

• For every rasterized map-element, perform depth test in target depth map.
  -> detect intersection.

Advantage:
Rasterization procedure can be very fast, due to regular grid transform.

Difficulties:
To establish spatial relationships and references
Distance maps (result)

Performance:

Mapsize = 900x600 = 0.54 Mio. voxels

Time for transformation and test:
0.745 sec. (300Mhz PIII)

Comparison:
Analytical polygon intersection testing
(0.1 millisec. per test)
Approx. 2 hours 30 minutes.

Risk:
Discrete technique is subject to
aliasing artifacts

Can be reduced using high resolution
distance maps.
Collision Testing Strategies : Object-Object

1. Step: Static collision management:
   Q: Which object are of interest for collision detection?
   A: Usually predefined in system build phase

2. Step: Dynamic collision management
   Q: Which object pairs must considered for collision detection at current simulation time?
   M: Spatial occupancy matrix, selection techniques, dynamics prediction
   A: List of paired objects of interest which might collide

3. Step: Sorting out the true-negative pairs
   Q: Which pairs of objects do certainly not intersect?
   M: Bounding volume testing, reduced LOD testing in some cases
   A: Reduced list of paired objects of interest which might collide

4. Step: Find true-positive pairs
   Q: Which objects do actually intersect?
      At which spatial location do they intersect?
      Texture coordinates?
      Impact angle?
   M: The entire palette of analytical tests