Collision Detection on Point Clouds Using a 2.5+D Image-Based Approach

Rafael Kuffner dos Anjos
Inesc-ID
Av. Prof. Dr Anibal Cavaco Silva
Portugal 2744-016, Porto Salvo
rafaelkuffner@gmail.com

João Madeiras Pereira
IST/Inesc-ID
Rua Alves Redol, 9
Portugal 1000-029, Lisboa,
jap@inesc-id.pt

João Fradinho Oliveira
C3i/Inst. Politécnico de Portalegre
Praça do Município
Portugal 7300, Portalegre
joaofradinhooliveira@gmail.com

ABSTRACT

This work explores an alternative approach to the problem of collision detection using images instead of geometry to represent complex polygonal environments and buildings derived from laser scan data, used in an interactive navigation scenario. In a preprocessing step, models that are not point clouds, are sampled to create representative point clouds. Our algorithm then creates several 2.5+D maps in a given volume that stacked together form a 3D section of the world. We show that our new representation allows for realistic and fast collision queries with complex geometry such as stairs and that the algorithm is insensitive to the size of the input point cloud at run-time.

Keywords

Collision Detection, Point-based Graphics, Navigation

1. INTRODUCTION

Collision detection is normally a bottleneck in the visualization and interaction process, as collisions need to be checked at each frame. Traditionally, the more complicated and crowded is our scene, the more calculations need to be done, bringing the frame-rate down. Therefore the optimization of this process, gaining speed without losing quality in the simulation, is something that has been researched for years. Although several different techniques and approaches have been developed, and showed good performance in specific scenarios, these approaches rely on object topology information which is easily extracted from polygonal models. However with point cloud models, the classical approaches either will not work, or will have to heavily adapt to this specific scenario, compromising their optimizations.

Using images as an alternative way of executing the task of collision detection might just be the answer. Image-based techniques can have their precision easily controlled by the resolution of the used images, and the algorithms are completely independent of the object's topology and complexity at run-time. It does not matter whether an object has sharp features, round parts, or even whether it is a point cloud, as all we are dealing with is the object's image representation. Being a scalable and promising technique, Image-based collision detection seems to be a plausible alternative to the classical approaches.

Our approach focuses in a virtual reality navigation scenario, where the scene is derived from the real world via devices such as laser scanners, which tend to generate enormous point clouds. Also, the hardware at hand might not fit the basic requirements for most algorithms and techniques, a situation that commonly will happen in tourism hotspots, museums, or other places where we would like ordinary people to be able to interact with the system. The developed application enables them to control an avatar on a static environment, a point cloud or polygonal model.

The main contribution of our research is a new 3D world representation for environment and buildings which is completely image-based with information that enables realistic and robust collision detection with underlying complex geometry such as slopes and stairs. Slicing a given structure along the Z axis (Using Cartesian coordinates as illustrated in Figure 1); we create a discrete set of images containing height information about the surface, and possible collidable frontiers. It is a flexible format that is able to represent either point clouds or polygonal models. This representation allows us to perform collision detection with user chosen precision, and high scalability.

2. RELATED WORK

The problem of collision detection is present in several areas of research and applications, each of them having different concerns, requirements and desired results. This necessity has prompted the creation of several techniques that try to deliver these results using varied approaches, each one fitting to specific problems.

Feature based: Famous examples are the Lin-Canny algorithm [MLJC] and its more recent related algorithm, V-Clip [BMM], which keeps track of the closest features between two objects, deriving both the separation distance, and the vertices that have possibly already penetrated the other object.

Bounding Volume Hierarchies: Different volumes have been developed to envelop the basic features, such as Spheres [PMH], Oriented Bounding Boxes (OBB) [SGMCLDM], or Axis Aligned Bounding boxes (AABB) [TLTAM] [MJBJ] [XZYJK], each of them has its advantages over the others; Spheres are easier to fit, OBBs have faster pruning capabilities, and AABBs are quicker to update, therefore being very popular in deformable body simulation.

Also, different tree traversing and creation techniques [TLTAM] [SGMCLDM] have been developed to optimize these expensive operations, taking into account each specific kind of application.

Stochastic algorithms: Techniques that try to give a faster but less exact answer have been developed, giving the developer the option to trade accuracy in collisions with computing power. The technique based on randomly selected primitives, selects random pairs of features that are probable to collide, and calculates the distance between them. The local minima is kept for the next step and calculations are once again made. The exact collision pairs are derived with Lin-Canny [MLJC] feature based algorithm. With a similar idea, Kimmerle et. al [SKMNFF] have applied BVH's with lazy hierarchy updates and stochastic techniques to deformable objects and cloth, where not every bounding box is verified for collision, but it has a certain probability.

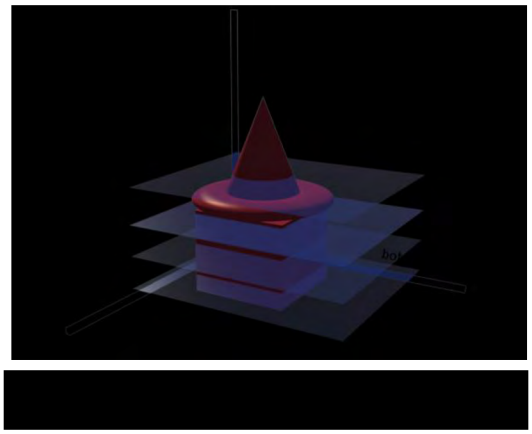


Image-based algorithms: These solutions commonly work with the projection of the objects, in contrast with previous techniques that work in object space, meaning that they are not dependent of the input structure, and as such are more suitable to point clouds. However to our knowledge they have not yet been applied to point clouds.

RECODE [GBWHS] and several other works [DKDP] [KMOOTK], [GBWSW] take advantage of the stencil buffer and perform collision detection on it by using object coordinates as masks, and thus detecting possible penetrations.

CULLIDE [NSRMLDM] uses occlusion queries only to detect potentially colliding objects, and then triangle intersection is made on the CPU. They render the bounding volumes of the objects in normal and reverse list storage order, and remove the objects that are fully visible in both passes, meaning that these objects are not involved in any collision.

Heidelberger et. al [BHMTMG] uses simple AABB's as bounding volumes for the objects in the scene. Potentially colliding objects are detected, and a LDI (Layered Depth Image [JSLR]) of the intersection volume between the two objects is created. That is, a volumetric representation of an object across a chosen axis. At each rendering step, as a polygon is projected into the LDI, the size of the intersubsection volume is computed. Faure et. al [FSJF] [JFHFCP] addresses not only collision detection, but also its response by using this same principle.

On a collision avoidance scenario, we want to predict an upcoming collision, and use this information to prevent it from happening. It is used mostly in artificial intelligence to control intelligent agents or robots. Loscos et. al [CLFTYC] represent the environment as patches to where the agent can or cannot go according to its occupation. It can be considered a basic but robust image based approach since it uses a 2D map to represent the environment.

Collision detection on point clouds: Algorithms using feature based techniques, bounding volumes, and spatial subdivision have been developed. Klein and Zachmann [JKGZ] create bounding volumes on groups of points so collision detection can be normally applied. Figueiredo et. al [MJBJ] uses spatial subdivision to group points in the same voxel, and BVHs to perform collision detection. The main issue while dealing with point clouds is the absence of closed surfaces and object boundaries. Ordinary BVH or stochastic techniques have to heavily adapt to this scenario, normally leading to not so efficient hierarchies. Feature-based techniques that work at vertex level are not scalable enough to be suited to these scenarios, since point clouds are normally massive data sets. Image-based techniques have the

```
for all points p in m do
    s ← floor((abs(z − z_min)) / σ)
    red ← (abs(z − z_min) mod σ) / σ
    red_old ← cube[x_screen][y_screen][s]
    if (abs(red_old − red)) / σ > σϵ then
        cube[x_screen][y_screen][s] ← 1
        p.color(red, 1, 1)
        if red_old > red then
            p.z ← p.z + (red_old − red) * σ + ϵ_2
        end if
    else
        cube[x_screen][y_screen][s] ← red
        p.color(red, 1, 0.1)
    end if
end for
```

disadvantage of sometimes requiring special graphic card capabilities, but only for some implementations. Overall, their properties make them the best suited for the proposed scenario of the tourism kiosk.

For further information on collision detection and avoidance techniques we suggest the following surveys: [NMAS] [SKTGKICB] [MT].

3. IMPLEMENTATION

Image-based algorithms that have been presented in the community ([NSRMLDM] [GBWHS] [DKDP] [NBJM] [JFHFCP] [FSJF]) perform very well in various kinds of scenarios, but some features of our set scenario (described on Section 1) make them hard or impossible to be applied (e.g. our data is unstructured, not all objects are closed or convex).

Our work extends the idea of a 2.5D map presented on the work of Loscos et. al [CLFTYC] by using enhanced height-maps, where the pixel color not only represents the height on that point, but also contains obstacle information, while at the same time overcoming the limitations of only supporting single floor environments. We call these enhanced maps, 2.5+D maps. Instead of having just one height map, we create a series of enhanced maps along intervals sized σ on the z axis, thus enabling the storage of more than a single z value for each (x, y) pair. Unlike LDI's, our representation does not combine several images into a volumetric representation, but separates each slice into a single image so we can easily perform memory management, and apply image comparison and compression techniques to have a better performance. Using the color of each pixel as a representation of a voxel, we write height information on the red channel, and identify obstacles on the blue channel. By adding these variables, we can determine not only the height level where the avatar should be standing, but also if he is colliding with any obstacle in several different heights.

Slices Creation

The creation of this representation is executed in a pre-processing stage, which is composed of several steps that must be performed from the input of the model until the actual rendering to create the snapshots that will be the used as collision maps. These slices are created as following. The camera is first set up according to the previously calculated bounding boxes of the input model on an orthogonal projection. After rendering that projection, a snapshot sized σ is created and saved into the disk for further use. The camera then is moved along the z axis, and the process is repeated until the whole extension of the model has been rendered into images. A visual representation of the described variables along with the slices computed on an example model can be seen on Figure 1 and two real slices can be seen on Figure 4.

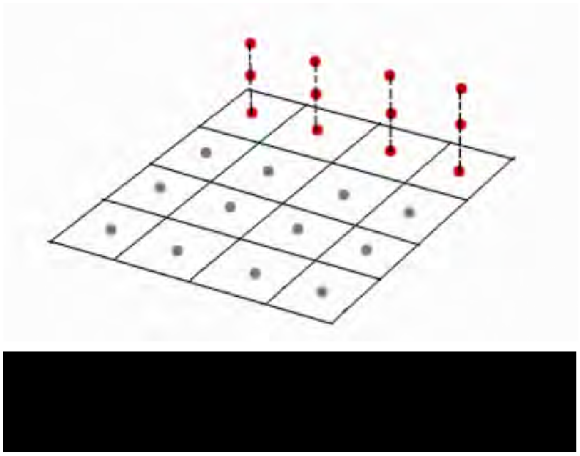
Polygonal Model Oversampling

We aim for a solution that accepts both polygonal models and point clouds. However these representations are inherently different and cannot be processed initially in the same way. Hence we created a solution that approximates the polygon models with a synthetic point cloud thus enabling later steps to be processed in the same way. We apply a simple oversampling operation that operates at triangle level transforming a polygonal model into a point cloud with a user-choice level of precision. After oversampling and discarding the polygons, the rendering of the produced point cloud has exactly the same shape and fill as if rendering the polygonal representation to the height map.

Information Processing and Encoding

Since all of our collision information will be written on collision maps as colors, we must assign each point on the point cloud a color representing its collision information. This will not replace the original color of that point in question. When writing these points on the output image, each pixel will represent a voxel sized (t, t, σ) on object space. So the painted color on that pixel will represent all the points contained in that volume. The algorithm on Figure 2 performs both the operation of height map information encoding, and obstacle detection. We define σ as $3t$, so as to ensure that one has more than one point on each slice, to properly perform the obstacle detection, as will be described later.

The first operation is executed as follows: We calculate the difference between the current point z coordinate and the model's lower bounding box z_{min}, and apply the modulo operator with σ. This remainder r represents the points z coordinate on an



interval $[0, \sigma]$. To be used as a color value, this difference must belong to the interval [0,1], so we calculate $^{r}/_{\sigma}$ thus deriving finally the red channel value. The simplified formula is given by $red = \frac{abs(z - z_{min}) mod\, \sigma}{\sigma}$

As navigation on a real-world scenario is normally performed horizontally on the xy plane, we classify an obstacle as a surface that is close to being perpendicular to xy, parallel to zy. So our obstacle collision technique simply estimates how parallel to the z axis a certain surface is. Figure 3 illustrates how potential obstacles and floor are classified using the coloring algorithm (Figure 2). Points lined up vertically on the same pixel most likely belong to a vertical surface. Diagonal structures like ramps are climbable up to a certain degree. The closer to a wall they are, the greater the probability is that its points are considered to be obstacles.

In order to keep track of point information that will be stored in the slices we use an auxiliary structure, a 3D array $cube[w][h][\sigma]$, after processing each point, we keep its color value on the position of the array representing the voxel on object space from where it came from. If there is already a stored value in this voxel, the difference between both red values is calculated, and transformed into an object-space distance $\frac{abs(red_{old} - red)}{\sigma}$.

If this difference is bigger than a certain small percentage ε (e.g. 7%) of the size σ of the slice, we assume that the points are vertically aligned, belonging to a vertical surface. These points are marked on their blue channel with the value 1, and we slightly increase its z coordinate so that the point is not occluded when rendering the maps. Typical output slices produced in the pre-processing stage can be seen in Figure 4, an office environment, where the floor has been correctly labeled as green, and all the walls are labeled as white or light blue.

Collision Detection

The developed representation provides us with enough information to perform quick collision detection on the navigation scenario given on section 1 where we consider point clouds as a viable input. In the accompanying video we show precise collision detection between rigid bodies and point clouds.

We divide the task of collision detection into two steps: a first step, that we call Broad phase, where we verify the occurrence of collisions between any objects in the scene, and a second step called narrow phase, where we perform collision response.

3.1.1 Broad Phase and Collision Detection

This task consists on identifying possible collisions between all objects on the scene. By representing the avatar that will be navigating on the environment by an Axis Aligned Bounding Box (AABB), we first calculate its size in pixels by calculating $pix_x = \frac{size_x}{t}$ and $pix_y = \frac{size_y}{t}$, where threshold t is calculated as the pixel size. This will be the number of pixels checked for collision on each slice, around the center of the avatar. If any checked pixel is not black, we mark the object as colliding, and they will be further processed in a narrow phase.

The only images we will need to load into the memory at the same time in order to perform collision detection between the given avatar and the environment are the ones located between $slice_0 = \frac{zp_{min} + zp - z_{min}}{\sigma}$ and $slice_n = \frac{zp_{max} + zp - z_{min}}{\sigma}$, where zp represents the z coordinate of the avatar. These slices contain collision detection information about the location where the pawn currently is.

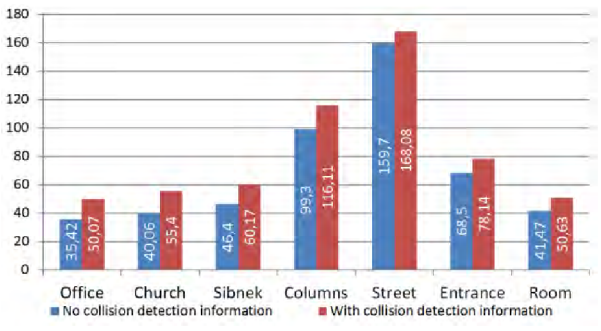
New slices that are needed, are loaded into memory until a user defined constant of number of slices (n_{slices}) is reached. New slices beyond this point, replace an already loaded slice that has the furthest z value from the avatar's own z value, meaning it is not needed at this point of the execution. In practice we found that six slices covered well the avatar's

potential waist, shoulders, head, knees, and feet collisions with the scene.

3.1.2 Narrow Phase and Collision Response

In our current version, the sole purpose of our collision response algorithm was to simply avoid objects from overlapping, and provide a basic navigation experience on the given environment. Any other more complex technique could be applied here, but this simple solution fulfills the requirements for our navigation scenario. We focused on an efficient broad phase algorithm, and a flexible representation so we could apply any chosen image-based technique on the narrow phase. This was achieved with a straightforward extension of our broad-phase algorithm, by applying the concepts of collision response from height maps, and collision avoidance [CLFTYC]. Instead of simply returning true when we find pixels that are not black, we gather information for collision response each time we find colored pixels. Pixels with the blue channel set to 1 always represent an obstacle, except on applications where we want to enable the avatar to climb small obstacles, as the agents from Loscos et.al [CLFTYC]. On these situations, we may ignore these pixels up until the height we want to be considered as climbable. As our avatar moves on fixed length steps, and each time it collides we correct it to the place it was on the previous check, we thus ensure that the avatar is always on a valid position. We apply this (x, y) correction each time an obstacle pixel is found until all the pixels representing the avatar's bounding box are verified.

Height is defined exactly as it is when precomputing height maps. By multiplying the coded height information on the red channel by σ and adding the z base coordinate of the given slice, we have precise information about a given point's height. Collision response can be made by setting the final height to the average height of the points on the base of the bounding box, or by the adopted strategy, the maximum value. Here we also check for surface height values from the first slice until the height we want to consider as climbable.



The complexity of this operation is exactly $O(pix_x * pix_y * s)$ where pix_x and pix_y are the number of pixels ocuppied by the base of the avatar and s is the number of slices encompassed by the avatar's height. We point however, that these checks are already performed in the broad phase, and hence can be re-used in the narrow phase without adding any extra operations.

4. EXPERIMENTAL RESULTS

We have implemented the whole algorithm using OpenGL 2.1.2, C and the OpenGL Utility Toolkit

(GLUT) to deal with user input and the base application loop management. The platform used for tests was a computer with an Intel core 2 Duo CPU at 2 GHz with 2GB of RAM, a NVIDIA GeForce 9400 adapter, running Microsoft Windows Seven x86.

Table 2 shows the time taken on the whole preprocessing stage for each model and configuration. Polygon sampling during the preprocessing of polygonal models is clearly the most computationally intensive task in the pipeline, as the two higher complexity point cloud models (Entrance and Room as seen on Table 1 and Figure 7) that did not require sampling had a faster performance. In the preprocessing phase the increase on processing time with point clouds is linear to point complexity. This linear growth is expected since each point must be checked for coloring once, and also for common input processing tasks such as file reading and display list creation.

Regarding the results of the overall memory cost specifically in the preprocessing phase, Table 2 shows that memory scales almost linearly according to the input size of the point cloud (Entrance versus Room in Table 1). This memory is needed temporarily for allocating the auxiliary 3D array for obstacle detection in the slice creation phase. Similarly, tests have shown that this memory consumption also grows linearly with the number of points produced in the polygon sampling phase.

During the application runtime, the average memory consumption varies according to the number of loaded slices into RAM, and according to the size of the loaded model used for rendering (Figure 5 and Table 1). On a 700x700 resolution, the peak minimal value found in any model we experimented was 50,07MB (Office) and the peak maximum 168,08MB (Street), with 6 slices loaded in memory. Table 2 shows how much memory the application consumes when only rendering the models and the total memory with collision detection, while having 6 slices loaded in memory. By controlling n_{slices} we can avoid this value from going over the memory we wish the

Model	Type	Original Complexity	Details
Office	Polygonal	17.353 pts.	Office environment with cubicles and hallways
Church	Polygonal	26.721 pts.	Simple church with stairs and columns
Sibenik	Polygonal	47.658 pts.	Cathedral of St. James on Sibenik, Croatia
Columns	Polygonal	143.591 pts.	Big environment with localized complexity.
Room	3D laser scan	271.731 pts.	3D Scan of a room with chairs and several objects.
Street	Polygonal	281.169 pts.	Outside street environment with an irregular floor, houses, and several objects
Entrance	3D laser scan	580.062 pts.	Entrance of the Batalha monastery in Portugal.

application to consume, since all other memory required for the application is for tasks unrelated to collision detection.

We were interested in studying the direct behavior of our algorithm and did not wish to mask performance with asynchronous I/O threads. Results on collision detection have been verified through establishing a fixed route to navigate with the pawn avatar where it goes through different situations and scenarios, such as "climbing" single steps, "traversing" planar areas, going "near" cylindrical or square shaped columns and "falling" from a height. Whilst the number of created collision maps for a scene can affect collision results, the actual number of buffer slices n_{slices} will only affect potentially the performance, as the system checks all slices at the avatar´s current height. Tests on Cathedral 700x700 with 130 slices and n_{slices} set to 10 have showed us that reading from the disk at runtime has a bigger impact on efficiency than storing a higher number of images on commodity RAM. For instance, when using a maximum buffer of 10 slices and reading a high resolution image from the disk, we notice a sudden drop in frame-rate (Figure 6), and this can also be noticed when the pawn falls from a higher structure, and needs to rapidly load a series of maps on his way to the ground. By increasing n_{slices} to 16 on this scenario, we ensure the storage of enough slices to represent the ground floor and the platform on top of the steps (Experiment B in the accompanying video). Little difference was noticed on memory load (5,33MB), and the interaction was much smoother.

Results also show that our algorithm did not affect in any way the rendering speed of the interactive application. Figure 6 shows that the frame-rate was nearly identical in the same scenario with and without collision detection using 16 slices.

Although we did not aim for high precision on collision response, our technique has presented precise results on different resolutions. We note that point clouds are themselves approximations to surfaces, and as such a collision amongst points is an unlikely event, Figueiredo et. al use the average closest point to point distance divided by two to establish a conservative bounding box around each point, which can generate false collisions when close. With our approach, instead of a box, we use the pixels of a zoomed out view which is also conservative. Precision results with the different algorithms were verified empirically by changing the

Model	Time (s)	Total Complexity	Memory
Office	41,23	9.349.585 pts	610,3 MB
Church	58,14	6.475.125 pts	536,58 MB
Sibenik	78,87	5.199.093 pts	532,48 MB
Columns	42,92	2.612.303 pts	241,66 MB
Street	114,75	7.142.361 pts	598,02 MB
Entrance	13,9	580.062 pts.	122,88 MB
Room	6,96	271.731 pts.	67,58 MB

	Image-based (700x700)	BVH Oct 4096
Frame-rate	30 fps	16 to 30 fps
Total Memory	143.36 MB	225,44 MB
Pre. proc. time	13.9 s	1500 s

color of the avatar when a collision occurs. We found that using collision maps with a resolution of 700x700 enabled one to lower the number of false collisions from other methods when close to obstacles.

Floor collision has been performed perfectly in all resolutions, since its precision is defined by the rgb value of the point. Collisions with obstacles are more affected by resolution as is to be expected, since we rely on pixel finesse to precisely know the position of a given surface. Tests on office and street (Figure 7) have showed the same errors of small object interference or fake collisions due to diffuse information about object boundaries. These are more noticeable on the interactions with round objects on Street (Figure 7) where we can notice the aliasing creating invisible square barriers around a perfectly round object.

Table 3 compares our technique with the work from Figueiredo et. al [MJBJ], which has been tested on one of our experimental scenarios (Figure 7), the walkthrough in the point cloud of the Batalha Monastery (Experiment A in the accompanying video, 700x700 resolution n_{slices}set to 10), on a machine with a slightly faster processing speed and RAM than the one used for our walkthroughs. We compared our results with their best performance alternative, that bases the surface partition on 4096 cells of the octree.

Frame-rate was disturbed during the collision detection process on the R-tree approach, while it remained steady at the 30 fps during the whole execution of our application. Also, the image-based technique has required much less memory to be executed, even with a significantly high number of slices loaded in memory. The biggest difference is in the pre-processing times. Our approach was executed 107 times faster than the BVH approach. The pre-processing stage must only be performed once for each configuration, since the representation is written and loaded to the hard-drive for further interactions.

As stated in section 2 the research on point cloud collision detection is recent, and non-existent regarding image-based techniques. Our pioneer solution has presented excellent results, not only performing better than other works on point clouds published in the scientific community, but also being flexible enough to be applied on models from CAD, or combined with precise collision response techniques. Our technique can be performed with our representation on any computer that can render the input model at an acceptable frame-rate, without requiring much processing from the CPU or GPUs.

5. CONCLUSION AND FUTURE WORK

A new image-based environment representation for collision detection has been presented, using 2.5+D slices of an environment or buildings across the z axis. These images contain at each pixel, information about a given voxel, representing i's contents with colors. Height map information is stored on the red channel, and obstacles are indicated on the blue channel. This allows us to have a Broad phase collision detection stage that is performed with high efficiency and scalability, where the user can choose the precision according to the computing power at hand by simply adjusting the resolution of the produced images. Point clouds and polygonal models are ubiquitously processed, making our approach currently the best suited for fast interaction with massive laser-scan data. This work fills a gap in the area of collision detection, exploring a scenario that has been receiving more attention recently.

Future Work

Using graphic card capabilities such as the stencil buffer for broad-phase collision detection and vertex shaders for point coloring could greatly speed up these operations, and also, calculations would be moved to the GPU, taking away some work from the CPU. Applying this representation of environments also on objects of the scene, or even applying it to the avatars we're using on the interaction, could present interesting results. Using non-uniform resolution images on environments where we do not have a uniform complexity, would also help us achieve more precision on our narrow phase, or on these presented situations.

Image comparison techniques and compression can also be applied to the generated images in order to decrease the number of times we need to load a slice, and also the number of collision detection checks we must do. In several man-made structures such as buildings, many slices tend to be identical; intra-slice compression also presents itself as an interesting avenue of research.

6. ACKNOWLEDGMENTS

We would like to thank Artescan for the point clouds provided.

7. REFERENCES

[BHMTMG] Bruno Heidelberger, Matthias Teschner, and Markus Gross, Real-Time Volumetric Intersections of Deforming Objects, Proceedings of Vision, Modeling, and Visualization 2003,461-468,2003

[BMM] Brian Mirtich, V-clip: fast and robust polyhedral collision detection, ACM Trans. Graph., 17:177--208, July 1998.

[CLFTYC] Céline Loscos, Franco Tecchia, and Yiorgos Chrysanthou, Real-time shadows for animated crowds in virtual cities, In Proceedings of the ACM symposium on Virtual reality software and technology, VRST '01, pages 85--92, New York, NY, USA, 2001. ACM.

[DKDP] Dave Knott and Dinesh K. Pai, Cinder - collision and interference detection in real-time using graphics hardware, Proceedings of Graphics Interface, pages 73--80, 2003.

[FSJF] François Faure, Sébastien Barbier, Jérémie Allard, and Florent Falipou, Image-based collision detection and response between arbitrary volume objects, In Proceedings of the 2008 ACM SIGGRAPH/Eurographics Symposium on Computer Animation, SCA '08, pages 155--162, Aire-la-Ville, Switzerland, Switzerland, 2008. Eurographics Association.

[GBWHS] G. Baciu, Wingo Sai-Keung Wong, and Hanqiu Sun, Recode: an image-based collision detection algorithm, In Computer Graphics and Applications, 1998. Pacific Graphics '98. Sixth Pacific Conference on, pages 125 --133, oct 1998.

[GBWSW] George Baciu and Wingo Sai-Keung Wong, Hardware-assisted self collision for deformable surfaces, Proceedings of the ACM symposium on Virtual reality software and technology, 2002.

[JFHFCP] Jérémie Allard, François Faure, Hadrien Courtecuisse, Florent Falipou, Christian Duriez, and Paul G. Kry, Volume contact constraints at arbitrary resolution, ACM Trans. Graph., 29:82:1--82:10, July 2010.

[JKGZ] Jan Klein and Gabriel Zachmann, Point cloud collision detection, Computer Graphics Forum, 23(3):567--576, 2004.

[JSLR] Jonathan Shade, Steven Gortler, Li wei He, and Richard Szeliski, Layered depth images, Proceedings of the 25th annual conference on Computer graphics and interactive techniques, 1998.

[KMOOTK] Karol Myszokowski, Oleg G. Okunev, and Tosiyasu L. kunii, Fast collision detection between complex solids using rasterizing graphics hardware, The Visual Computer, 11(9):497 -- 512, 1995.

[MJBJ] Mauro Figueiredo, João Oliveira, Bruno Araújo, and João Pereira, An efficient collision detection algorithm for point cloud models, 20th International conference on Computer Graphics and Vision, 2010.

[MLJC] M.C. Lin and J.F. Canny, A fast algorithm for incremental distance calculation, In Robotics and Automation, 1991. Proceedings., 1991 IEEE International Conference on, pages 1008 --1014 vol.2, apr 1991.

[MT] M. Teschner, S. Kimmerle, G. Zachmann, B. Heidelberger, Laks Raghupathi, A. Fuhrmann, Marie-Paule Cani, François Faure, N. Magnetat-Thalmann, and W. Strasser, Collision detection for deformable objects, Computer Graphics Forum,24(1):61--81,2005

[NBJM] Niels Boldt and Jonas Meyer, Self-intersections with cullide, Eurographics, 23(3), 2005.

[NMAS] Noralizatul Azma Bt Mustapha Abdullah, Abdullah Bin Bade, and Sarudin Kari, A review of collision avoidance technique for crowd simulation, 2009 International Conference on Information and Multimedia Technology, (2004):388--392, 2009.

[NSRMLDM] Naga K. Govindaraju, Stephane Redon, Ming C. Lin, and Dinesh Manocha, Cullide: interactive collision detection between complex models in large environments using graphics hardware, In Proceedings of the ACM SIGGRAPH/EUROGRAPHICS conference on Graphics hardware, HWWS '03, pages 25--32, Aire-la-Ville, Switzerland, Switzerland, 2003. Eurographics Association.

[PMH] Philip M. Hubbard, Approximating polyhedra with spheres for time-critical collision detection, ACM Transactions on Graphics, 15(3):179--210, July 1996.

[SGMCLDM] S. Gottschalk, M. C. Lin, and D. Manocha. 1996. OBBTree: a hierarchical structure for rapid interference detection. In Proceedings of the 23rd annual conference on Computer graphics and interactive techniques (SIGGRAPH '96). ACM, New York, NY, USA, 171-180.

[SKMNFF] Stephan Kimmerle, Matthieu Nesme, and François Faure, Hierarchy Accelerated Stochastic Collision Detection, In 9th International Workshop on Vision, Modeling, and Visualization, VMV 2004, pages 307-312, Stanford, California, États-Unis, November 2004.

[SKTGKICB] S. Kockara, T. Halic, K. Iqbal, C. Bayrak, and Richard Rowe, Collision detection: A survey, In Systems, Man and Cybernetics, 2007. ISIC. IEEE International Conference on, pages 4046 --4051, oct. 2007.

[TLTAM] Thomas Larsson and Tomas Akenine-Möller, Collision detection for continuously deforming bodies, Eurographics 2001.

[XZYJK] Xinyu Zhang and Y.J. Kim, Interactive collision detection for deformable models using streaming aabbs, Visualization and Computer Graphics, IEEE Transactions on, 13(2):318 --329, march-april 2007.