Virtual and Real Object Collisions in a Merged Environment

by

Daniel G. Aliaga

A Thesis submitted to the faculty of the University of North Carolina at Chapel Hill in partial fulfillment of the requirements for the degree of Master of Science in the Department of Computer Science.

Chapel Hill

1993

Approved by:

_____ Advisor: Henry Fuchs

_____ Reader: Gary Bishop

_____ Reader: Dinesh Manocha

DANIEL G. ALIAGA. Virtual and Real Object Collisions in a Merged Environment.

ABSTRACT

See-through head-mounted display capability is becoming an important part of Virtual Environment applications. In such applications, it may be desirable to model the physical behavior of the virtual objects and their interaction with the real objects. This thesis describes a software system that provides interactive collision detection and collision response for see-through head-mounted displays. The system employs a static model of the real world environment and allows for arbitrary convex virtual objects to be placed in the environment. The user may control the positions and velocities of the virtual objects. An approximately constant time collision detection algorithm and a Newtonian based single point contact collision response is used to model the apparent physical interaction of the virtual and real objects for moderately complex environments.

PREFACE

I would like to thank my advisors, Henry Fuchs, Gary Bishop and Dinesh Manocha for their patience and wisdom. I would like to thank the additional people who have helped me during the writing of this document as well as in the development of the thesis. Especially, Amitabh Varshney for answering my late night questions, Mark Mine for the design of the virtual basketball court, Anselmo Lastra for answering my questions regarding Pixel-Planes 5, Jack Goldfeather and John Halton for answering my multiple questions and Rich Holloway for assisting with the see-through head-mounted display hardware. I would also like to thank Marc Olano, Anselmo Lastra and Amitabh Varshney for reading the earlier versions of this document and to all others in the glab who have lent me a helping hand.

TABLE OF CONTENTS

Section **Page**

LIST OF TABLES

LIST OF FIGURES

LIST OF ABBREVIATIONS

BSP	Binary Space Partition (Tree)
GP	Graphics Processor
HMD	Head-Mounted Display
HP	Hewlett-Packard Company
MGP	Master Graphics Processor
MIMD	Multiple-Instruction-Multiple-Data (Computer)
PHIGS	Programmer's Hierarchical Interactive Graphics System
PPHIGS	Pixel-Planes PHIGS
RGB	Red-Green-Blue
SGP	Synchronization Graphics Processor
SIMD	Single-Instruction-Multiple-Data (Computer)
VROC	Virtual and Real Object Collisions (System)

Chapter 1

Introduction

1.1 Thesis Statement

It is possible to provide, using computational power readily available today, a see-through head-mounted display system with interactive collision detection and collision response for moderately complex environments containing both virtual (computer generated) objects and real objects.

1.2 Motivation

Among the original applications considered part of Virtual Environments are head-mounted display (HMD) systems. A typical HMD setup consists of a powerful graphics engine, one or more tracking devices and a head-mounted display. The user will experience the illusion of being in a (synthetic) world where the images seen are generated by a computer program. By using optical lenses or video camera technology it is also possible to present the user with images of the virtual environment and the real environment simultaneously. A helmet with such characteristics is called a see-through head-mounted display.

See-through capability opens up an even larger number of potential applications. An historically early example of the merging of the virtual environment with the real environment is a helicopter pilot's helmet where the pilot can see information about the helicopter's orientation, speed and location as well as a cross-bar that could be tracking an enemy all superimposed on the pilot's view of the real environment. Bajura et al. [1992] have experimented with superimposing ultrasound images of a fetus registered in place over a pregnant women's womb. Soon we could hope to see applications such as an architectural design system. Such a system could allow for an architect to make actual-size modifications to an existing building or for a home-owner to decorate an empty house. A similar application, could allow children to design and build a virtual toy which could be used simultaneously with real toys.

The real world, obviously, correctly models our physical environment, such as the effects of gravity, friction and collisions. In future applications of merged virtual and real environments, it may become useful to model the laws of nature; otherwise, the interaction of the two worlds may not be convincing at all. A significant amount of work must be done for a virtual environment to convincingly simulate such properties. Consider an office environment where the user has a virtual notepad. It would not be convincing if when the notepad is placed on the table, it apparently falls through the table. Similarly, in the previous example of a home-owner decorating an empty house, the home-owner might desire the addition of a sliding door or venetian blinds. These virtual additions should properly interact with the surrounding (real) house. I have solved part of this problem by designing an integrated software system, Virtual and Real Object Collisions (VROC), which provides interactive collision detection between virtual objects and models of real objects and collision responses for the virtual objects of a moderately complex merged environment of virtual and real objects.

1.3 VROC System

The VROC system uses an optical see-through head-mounted display together with a powerful graphics engine (also developed here at UNC-CH) to present to the user virtual objects superimposed on a real environment. The user provides a model of the static real environment which describes the position and sizes of the real objects. The user then creates any number of virtual objects. A hand-held tracker is provided with which the user can grab and control the linear and angular velocities of the virtual objects. The system constantly performs collision detection and computes a Newtonian based collision response to model the interaction (i.e. collisions) of virtual and real objects, as well as the interaction among the virtual objects themselves.

Chapter 2

Previous Work

2.1 Collision Detection

2.1.1 Criteria

Collision detection is a difficult problem to solve. It is inherently a computationally expensive operation. Over the last decade, multiple approaches have been developed for collision detection. The general problem could be stated as:

1. Given a set of objects determine if any of them intersect;
2. Determine which are the colliding objects and the collision point or surface.

No one collision detection algorithm can be said to be the optimal solution. One must take into consideration the application being designed and determine what information can be made available to the collision detection algorithm and what information will be needed from the collision detection algorithm. These two factors are mainly determined by how the objects are represented and on the action(s) to perform after the collision (the collision response).

2.1.2 Representations

The representation of the potentially colliding objects plays a decisive role in determining what geometrical information is available to the collision detection algorithm. A polyhedral representation is good since it provides for a simple way to model almost any object. On the other hand, collision detection operations on polyhedra are usually expensive. Functional representations, while not as versatile as polyhedra, do allow for fast collision computations. This dichotomy partitions the existing collision detection algorithms into at least two major categories:

(a) Algorithms applicable to polyhedral models:

Though it is possible to model almost any object with polyhedra, frequently it becomes necessary to decompose complex (concave) polyhedral objects into its convex components for collision detection purposes. Hahn [1988], Baraff [1989], Moore and Wilhelms [1988] assume (convex) polyhedral objects for their collision detection and collision response systems of rigid bodies. The general approach is to consider all possible intersections of vertices, edges and faces of polyhedral objects. The algorithm checks if the vertices of one object are contained in another object. Edge to edge intersection as well as face to face collisions must also be accounted for. Bounding boxes, among other techniques, can be used to improve performance; but, in general these algorithms are dependent on the polyhedral complexity of the objects and are usually expensive (O(vf), where v is the number of vertices and f is the number of faces).

Canny [1986] suggested another collision detection method for polyhedra moving among polyhedral obstacles. He approaches collision detection more as a collision avoidance problem and takes collisions into higher dimensions (configuration space). A set of multidimensional constraints is constructed which he reduces to a univariate polynomial. Cameron [1990] explored the possibilities of 4-space intersection testing. His method can be best understood by a 2-space collision detection analogy. The path a 2-space object takes over time forms a prism in 3-space. Two objects intersect, if and only if their corresponding prisms in 3-space do.

(b) Algorithms applicable to functional (implicit) models:

The use of functional models makes the process of creating a model of an arbitrary object more difficult. In order to overcome this, a significant amount of work has been done to provide mechanisms to expand the range of objects that can be easily constructed with functional models (also called implicit functions). Terzopoulos and Metaxas [1991] describe deformable superquadrics and how they can be made to fit 3D data. Sclaroff and Pentland [1991] describe a method of generalizing implicit functions. Terzopoulos and Witkin [1988], Pentland and Williams [1989] use similar implicit functions for physically based animations. Herzen et. al. [1990] suggested a method for time-dependent parametric surfaces. Most of the methods above allow for non-rigid objects. Non-rigid objects and their more complex collision responses are much easier to implement when using a functional representation. Implicit function collision checking can be done analytically or a hybrid polygonal approach can be used. For example, a simple approximate algorithm would use the superquadric's inside-outside function to check in O(v) time whether a vertex of object A is contained in object B, as opposed to the $O(v^2)$ approach that a conventional polygonal method would require (v is the number of vertices per object).

2.1.3 Large Scale Environments

For environments containing only a few objects, collision detection is in fact not a problem. Inefficient algorithms can be used and near real-time performance can be maintained. But as the number of objects increases (and potentially the complexity of each object), collision detection becomes very expensive.

Most collision detection algorithms compute the collision (if any) of a pair of objects. Hence an environment of n objects would typically require $O(n^2)$ collision checks. It is not unusual for n to be 100, 1,000 or even 10,000 objects; performing n^2 collision checks for every frame is far too computationally expensive. Typically, the number of actual collisions stays relatively small. Hence, the goal for an efficient paradigm for large scale environments is to have output sensitive algorithms. In other words, the total collision computation time depends not on the total number of objects, but on the number of actual collisions.

In order to discern which of the potential collision pairs will become actual collision pairs in the near future, some sort of subdivision or ordering of the potential collision pairs is needed. The subdivision can be based on space, time or some combination of space and time. Namely, based on the objects' positions it can be assumed that only the neighboring objects are potential collision candidates. If a bound is given on the maximum distance an object can travel over the next few frames, a potential collision radius can be computed for each object. For example, the objects can be hashed into the cells of a 3D grid. Each object need only be checked for collision with the objects of the neighboring cells. Unfortunately, if a large number of objects are moving, the objects might need to be rehashed into different cells at every frame. Alternate implementations of similar schemes use Binary Space Partition (BSP) trees, octtrees, non-inform 3D grids, etc [Pentland90b][Hahn88].

2.1.4 Dynamical Environments

If the objects exist in a dynamical environment (i.e. a world where object positions and orientations are constantly changing over time), the collision detection problem becomes more difficult. A key-frame approach can be used. Namely, at predetermined intervals, check the objects for penetration. For a small enough time step, the exact moment of collision must have occurred slightly before a penetration. Another approach to the collision detection problem of moving objects is to incorporate time as a parameter of the collision. Thus, instead of only computing the point (or surface) of collision, also determine the collision time based on the object's current velocities and accelerations.

With regards to what information will be needed from the collision detection algorithm, it is dependent almost entirely on the collision response algorithm that will be used. The collision response might be a simple reflectance vector calculation or the collision response might entail multiple computations in order to model the transfer of

momentum during a collision. In other cases, it suffices for the collision detection algorithm to signal when objects are almost in collision (as might be the case with objects formed from volume data) and thus not provide precise information about the collision.

Many algorithms have been formulated for collision detection. For the specific application of this system polyhedral objects, for the most part convex, sufficed. A rather new method, developed by Lin and Canny [1991,1992], gives approximately constant time collision detection between convex polyhedra moving over time. It is the algorithm used by the VROC system and will be described in Section 3.2.

2.2 Dynamics Simulations

The general problem could be stated as: given a set of objects model their physical behaviors over time. Many mathematical models exist that implement physical behaviors. But, since even a single aspect of physical behavior can be difficult to model, implementations usually only model a small number of physical behaviors and perhaps crudely approximate a few others. Wilhelms et al. [1988] present the general idea behind the use of physical simulations. Goldstein [1950] describes most of the issues of Classical Mechanics, while Baraff [1992] provides a good general overview of rigid body dynamics for 3D animations. It is hard to say which solutions are more "correct" than others. Some collision responses will construct a set of equations that describe a form of conservation of energy, other methods might place a temporary spring between the objects in contact and then based on the penetration depth, generate a large force for a small period of time (similar to an impulse force).

Li and Canny [1990] describe a model for rigid bodies with rolling constraint. Using the geometry of the rigid bodies they determine an admissible path between two contact configurations. No conservation of energy nor spring models are used. On the other hand, Baraff [1989] and Baraff [1990] give a formulation for contact forces between curved surfaces and objects in resting contact (though only point contact is modeled). Other methods, like those described by Moore and Wilhelms [1988] assume rigid bodies and model single point contact. They also allow for approximations of elasticity and friction. Hahn [1988] describes a general system for the dynamic interaction between rigid bodies. He presents models for elasticity, friction, rolling and sliding contacts. Other methods allow for objects to deform upon impact. For these methods, a quite complex analytical approach is taken. Pentland and Williams [1989] use vibration-mode ("modal") dynamics, a method of breaking down non-rigid dynamics into the sum of independent vibration modes. The ThingWorld System, developed at the Massachusetts Institute of Technology by Pentland et. al. [1990], allows for the creation of a virtual world where simple objects can collide and deform appropriately, through the use of modal dynamics. Witkin et. al. [1990] construct a complex physical environment by snapping together smaller simpler pieces (constraint equations) and then provide a system that tries to satisfy (solve) all the constraints.

The general problem of contact force determination is in fact quite difficult. Baraff [1989,1990] only describes methods to compute forces for point contact and curved surface contact. Bouma and Vanecek [1993] present a model for polyhedral contact that assumes there is no friction (thus can model each contact region by a finite number of points). The introduction of friction makes the problem much harder. In fact, Baraff [1993] presents various issues in computing contact forces for frictionless and frictional environments. He shows that the frictional consistency problem (the problem of deciding if a given configuration with dynamic friction is consistent) is NP-complete.

Pentland [1990b] states that the next major step we should take is to confront the problem of providing real-time physical behavior in a Virtual Environment. The computational complexity, as referred to by Pentland, of such systems is very high and does not scale linearly with the problem size. He states four areas that are critical to achieve complex real-time Virtual Environments: rendering, dynamics, collision detection and constraint satisfaction.

The VROC system provides simple solutions for the above areas (except for constraint satisfaction, which is not implemented) for moderately complex environments. The VROC system integrates collision detection, collision response and see-through head-mounted technology to present to the user an interactive environment of virtual and real objects. The following chapter will describe each component of the system and how they are integrated.

Chapter 3

Overview

3.1 Hardware Platform

3.1.1. Pixel-Planes 5

The VROC system is implemented on Pixel-Planes 5 [Fuchs89], though the original version was prototyped on a HP-750 TVRX-T4 workstation. Pixel-Planes 5 is a high-performance, scalable multicomputer for 3D graphics. Pixel-Planes 5 exploits screen subdivision and pixel parallelism to provide a platform for real-time algorithm research. Sufficient "front-end" for this level of performance is provided by a Multiple-Instruction-Multiple-Data (MIMD) array of general-purpose math-oriented processors (Intel's i860s). These general-purpose processors (referred to as Graphic Processors or GPs), the rendering units, the Frame Buffers and a host workstation are interconnected by a high-bandwidth ring network (8x20 MHz, 32 bits wide). A typical system has from 10 to 40 Graphic Processors and from 5 to 20 Renderers and is capable of generating over 2 million Phong-shaded triangles per second.

The host workstation runs UNIX and links Pixel-Planes 5 with other input devices. The host is used to load the GPs with the application executables stored on disk as well as provide general access to secondary storage. The host workstation can also run the control loop that sends commands to the GPs.

The Graphic Processors are fully programmable in the C language. Much of the system complexity is hidden by the machine's operating system (Ring Operating System); the programming model is therefore relatively simple. The code running on the GPs can perform application-defined computations. The application has at its disposal the high-bandwidth ring network for message passing among the GPs and the host workstation. The ring network is also used to send instructions to the multiple Pixel-Planes 5 Renderers.

The Renderers employ a novel Single-Instruction-Multiple-Data (SIMD) approach to graphic image generation. A GP sends a sequence of instructions to a Renderer's SIMD array of 128x128 processors. Each Renderer will typically receive rasterization instructions for a 128x128-pixel region of the screen. The Renderer

computes the pixel values from the primitive data (polygons, spheres, lines, etc.) and eventually block-transfers the final RGB values to the Frame Buffer. Once all 128x128-pixel regions of a frame have been computed, the frame is displayed.

The SIMD array of processors is not used for collision detection nor collision response purposes. It is possible to envision an implementation that uses the SIMD array processing power to evaluate some form of collision function on a per-pixel basis. The VROC system uses the multiple GPs instead.

3.1.2 PPHIGS Graphics Library

Pixel-Planes 5 may be programmed at multiple levels: by an application programmer, who simply desires a fast rendering platform with PHIGS+ style interface [van Dam88]; or by a system prototyper, who needs access to the GP's general-purpose (parallel) computing power but does not wish to worry about the actual rendering process (such as the VROC system); or by a programmer who wishes total access to the SIMD rendering processors and the GP's general-purpose (parallel) computing power.

A local variation of PHIGS+ (Pixel-Planes PHIGS or PPHIGS) [Ellsworth90] provides a high-level interface for users desiring portable code. PPHIGS makes the heterogeneous multi-processor hardware appear like a standard graphics system. The programmer's code, running on the host workstation, makes calls to the graphics library to build and modify a hierarchical database. The calls are converted to messages sent to the Master Graphic Processor (MGP) and the other GPs. In order to take advantage of the multiple processors, the database is distributed across the GPs in a way that balances the computational load, even in the presence of editing and changes in view. Every GP has a copy of the entire database hierarchy but only stores a portion of the primitives. A GP performs geometric transformations for the primitives it has and sends the corresponding rasterization instructions to the assigned Renderer. Thus, the major steps in the rendering process are:

1. The application program on the host workstation edits the database using the PPHIGS library calls. These changes are transmitted to the MGP (and the GPs).

2. The application requests a new frame. The host will send a message to the MGP, which relays it to the other GPs. This starts the frame processing as follows:

3. The GPs traverse the database, generating the Renderer commands for each primitive. These commands are placed in bins corresponding to each 128x128-pixel region of the screen (80 bins for a 1280x1024 image).

4. The GPs send the bins to the Renderers in an order determined by the MGP. The Renderers execute these commands and compute intermediate values.

5. Once all the bins have been processed, the Renderers compute the final pixel values and send the RGB pixel values to a Frame Buffer.

6. When a Frame Buffer has received all the regions, the Frame Buffer swaps banks and displays the newly-computed frame.

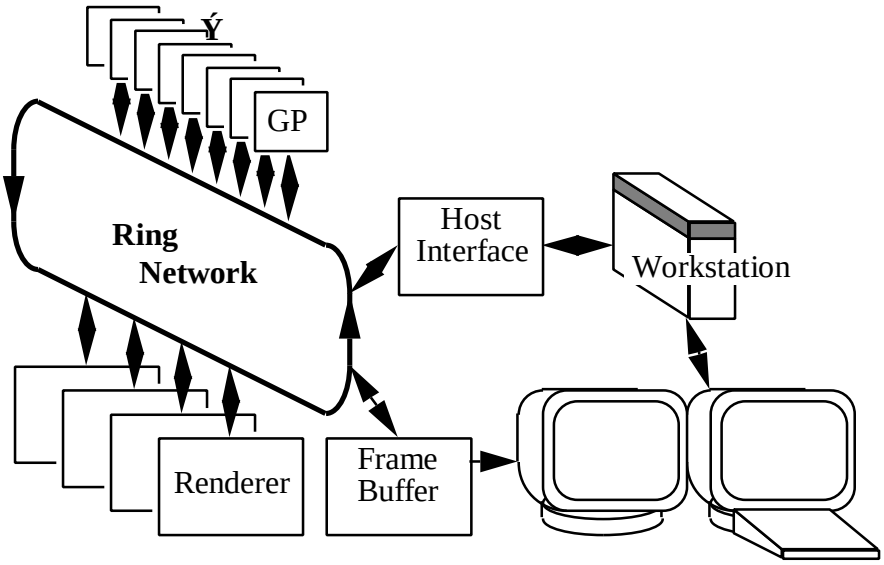


Figure 3.1: Pixel-Planes 5

3.1.3 See-through Head-Mounted Display Interface

The addition of the see-through head-mounted display does not interfere, in any significant way, with the standard rendering flow of Pixel-Planes 5 nor does it directly affect the VROC system collision computations. A non-head-mounted display application would have to specify the view matrix for each frame (or use the same one for multiple frames). The addition of the (see-through) head-mounted display simply requires a pair of view matrices (stereo image) to be recomputed based on the latest head-mounted tracking device information.

A typical HMD application of Pixel-Planes 5 uses the PPHIGS library as well as an additional set of libraries to interface the (see-through) head-mounted display and the multiple trackers (head-mounted tracker and optional hand-held tracker). The control loop on the host workstation receives position and orientation updates from the trackers. It then performs simple matrix transformations to compute the view for each of the "eyes" of the (see-through) head-mounted display. The resulting view matrices are used when instructing PPHIGS to compute the next frame for each eye.

3.1.4 VROC System

The VROC system is an application that uses the PPHIGS interface to Pixel-Planes 5. It creates a simple hierarchy and uses the PPHIGS callback mechanism to parallelize the computation of the object positions among the GPs.

In a standard PPHIGS application, the database stores multiple structures whose elements are either graphic primitives, state-changing commands, or calls to execute other structures. The PPHIGS callback mechanism allows for an element of a structure to be a callback, namely a function invocation. At the beginning of a new frame, each GP will traverse the entire database. When a GP encounters a callback element, the corresponding function is called. This function has access to the GP's memory and processing power and thus can modify or add any element to the display list. This function is the hook into the VROC system.

The VROC component on each GP handles a subset of the virtual objects in the environment as well as a subset of the collision object pairs used for the collision detection and collision response.

Sections 3.2-3.4 will describe the VROC system's algorithmic implementation. Section 3.5 will describe in more detail how the computations are distributed among the various GPs through the use of the aforementioned callbacks.

3.2 Collisions

3.2.1 Collision Detection Algorithm

In order to maintain the interactive performance of the system, selection of a fast collision detection method is essential. Lin and Canny [1991,1992] describe a method which integrates well with a dynamics system. A typical dynamics simulation advances through time by taking small time steps. The algorithm by Lin and Canny provides approximately constant time collision detection between convex polyhedra from one frame to another by assuming that the objects' positions and orientations will not drastically change from one frame to another.

The essentials behind the algorithm are rather simple. Given a pair of convex polyhedra, determine the closest features of the objects. For a three dimensional polygonal object, the features are vertices, edges and faces. Figure 3.2 shows the closest features between a cube and a tetrahedron.

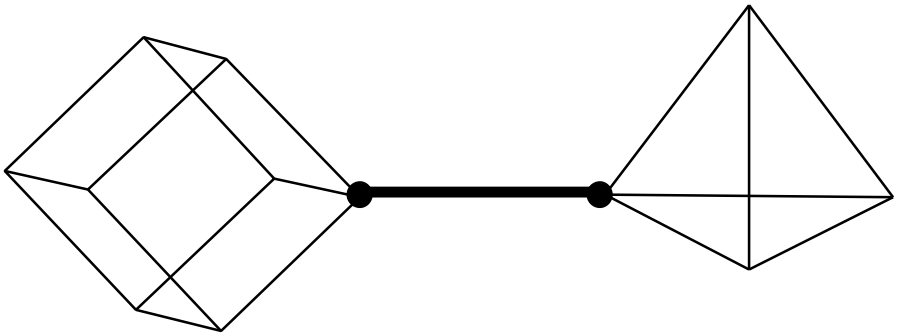


Figure 3.2: Closest features - example.

In the next frame, assuming a small enough time step, the objects will perhaps have rotated a few degrees and closed the distance between them. Figure 3.3 depicts what the configuration of the closest features might be during the next two frames.

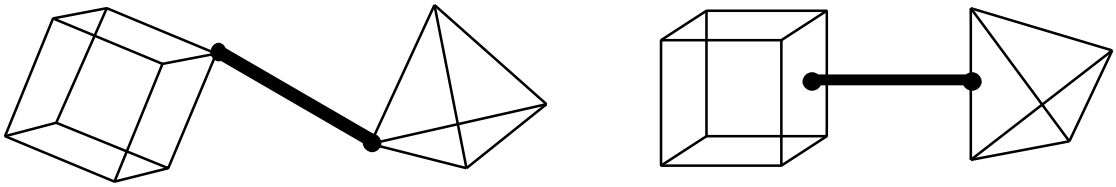


Figure 3.3: Closest features - possible next two frames.

The angular velocity of the objects caused the closest features to change, but only to the next adjacent feature; namely from a vertex to the face (or edge) adjacent to it. Given larger angular velocities, it might take a few more checks of the adjacent features. In any case, the algorithm provides approximately constant time distance checking. When the distance between the features is less than a tolerance value, the objects have collided.

Since spheres and cubes are such common objects, the collisions among them are treated as special cases. Namely sphere-sphere collisions and sphere-cube collisions require a simple distance check [Glassner90].

3.2.2 Collision Response Algorithm

Once two objects have collided, a response must be computed. The VROC system assumes that all objects are rigid and have nearly inelastic properties. Furthermore, only single point contact is modeled (since all objects are convex, this is generally the case).

The response is based on Hahn [1988] and Moore & Wilhelms [1988] work. The collision response computation assumes that the point of contact, velocities at

collision time and an orthogonal collision frame are provided. Then based on the conservation of linear and angular momentum, the new resulting velocities can be computed. The set of equations which describe this are:

$$m_1\bar{v}_1 = m_1v_1 + R$$
$$m_2\bar{v}_2 = m_2v_2 - R$$
$$I_1\bar{w}_1 = I_1w_1 + p_1 \times R$$
$$I_2\bar{w}_2 = I_2w_2 - p_2 \times R$$

The variables m, I, v, w describe each object's mass, inertia tensor matrix, linear velocity and angular velocity. The p vector is the relative vector from each object's center of mass to the point of contact and R is the impulse transfer vector. Each object has its own elasticity coefficient (value between 0 and 1). In order to simulate (slightly) elastic collisions, the R vector is scaled by the minimum of the two elasticity coefficients.

3.2.3 Time

The previous two sections outlined the collision detection and collision response algorithms. Now they must be combined to form the dynamics simulation. This implies that all the computations must be parametrized by time. The user must specify the time step to use to go from one frame to the next frame. The main problem with key-frame collision detection is that objects with large velocities might penetrate or even pass through each other in one frame transition. To prevent this, small enough internal time steps must be taken between frames. Namely, given a bound on the linear velocity it is easy to compute the maximum displacement any object can perform. By setting an appropriate collision distance (the distance at which two objects are considered to be in contact), it can be guaranteed that the objects will never penetrate or pass through each other in a single internal time step.

For example, if the maximum velocity is v_m and the collision distance is d, then $d/(2*v_m)$ is the maximum allowable time step. If the requested frame time step is larger, it must be subdivided into smaller internal time steps.

If at the end of an internal time step, the object pair is already in collision (penetration has occurred), a binary subdivision method through time is used to find the time of collision to within a certain tolerance. Furthermore, the instantaneous velocities and collision point are recomputed in order to obtain a more accurate collision response. The simulation must then restart at the collision time. Figure 3.4 shows 3 frame time steps. Each frame time step is divided into 4 internal time steps. A collision has occurred during the last internal time step of frame 2. Binary subdivision is used to find a better estimate of the collision time.

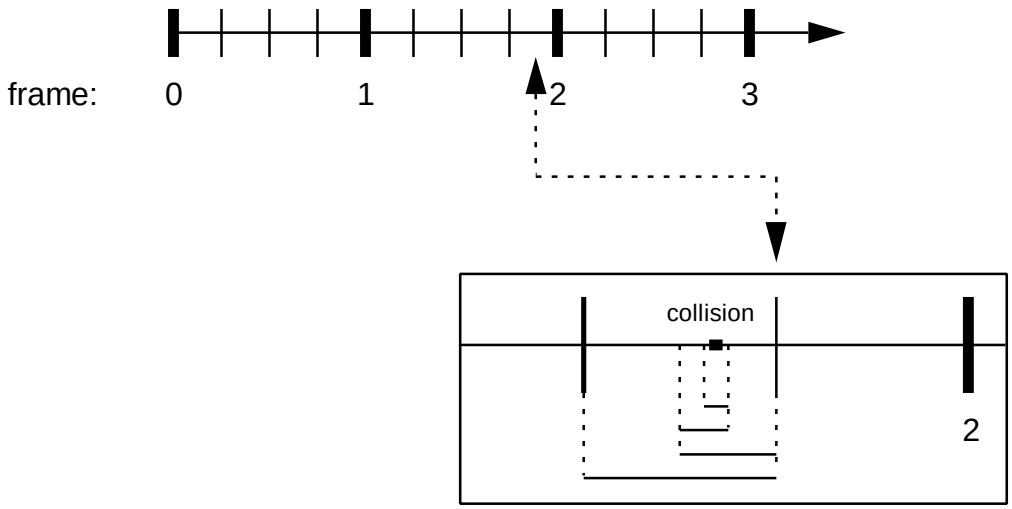


Figure 3.4: Subdivision of time steps, during non-collision and collision frames.

3.3 Optimizations

3.3.1 Scheduling scheme

3.3.1.1 Expected Collision Time

Since objects have continuous motion, it is possible to construct a sorted list of possible collision times. Given the distance between two objects, the bounds on the maximum linear velocity and linear acceleration, it is possible to predict the earliest time at which an object pair could collide.

The collision detection algorithm described in Section 3.2.1 provides the distances between the surfaces of the object pairs at no extra cost [Lin91]. Since objects have angular velocities as well as linear velocities, some additional work must be done in order to use the inter-object distance for prediction. For each object, there is an inner radius and an outer radius (the radius of the inscribed sphere and the radius of the circumscribing sphere). After subtracting from the inter-object distance the two differences between the inner- and outer-radius of each object, it is safe to use that distance for the prediction. Figure 3.5 illustrates this procedure. For non-moving objects, the difference between the radii can be considered zero, since the objects are convex and have no angular velocity.

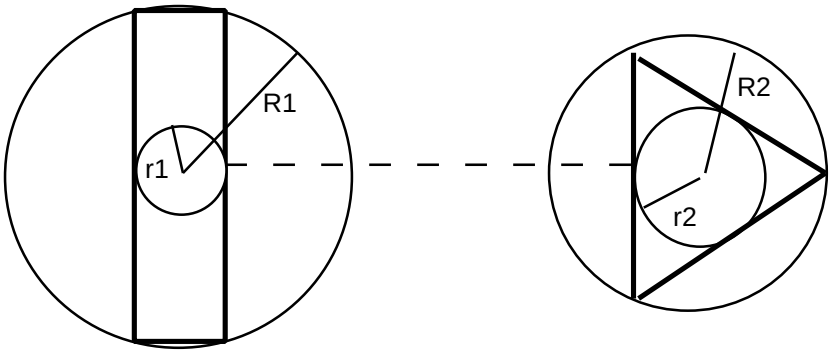


Figure 3.5: Inter-object distance, reducing the inter-object distance by $(R_2-r_2) + (R_1-r_1)$ ensures that the prediction will never be incorrect.

If the objects have no acceleration, only constant linear velocity, prediction is quite trivial. Specifically, by using the reduced inter-object distance d computed above and the maximal linear velocity component v_m, a collision cannot occur before current time plus d/v_m.

The more difficult case occurs when there is a linear acceleration component. This is the case of gravity, namely a constant force being applied to all objects in one direction. For an accurate prediction, it may be necessary to take into consideration the method by which the velocity is integrated. A Runge-Kutta integration method would provide a fairly accurate integration at the expense of computation time. A simpler Euler integration method would not provide such an accurate integration but will still produce the same effects one would expect from gravity at the cost of very little computation. Unfortunately, the inaccuracy of the latter integration is large enough that one must consider it when predicting the potential time of collision.

An Euler-type method for integrating the velocity and position would be:

$$v_{n+1} = v_n + a?\,t \qquad (1)$$

$$x_{n+1} = x_n + v_n?\,t \qquad (2)$$

If the initial velocity is v_0 and position is x_0, after n iterations, the velocity and position would be:

$$v_n = v_0 + na?\,t \quad (3)$$

$$x_n = x_0 + (v_0 + v_1 + \ldots + v_{n-1})?\,t \qquad (4)$$

Thus the general form that describes the distance that can be covered given the initial velocity and the time step, is:

$$d = \Delta t(nv_0 + (\sum_{i=0}^{n-1} i)a\Delta t) = \Delta t(nv_0 + \frac{n(n-1)}{2}a\Delta t) \qquad (5)$$

Hence, given an inter-object distance of d, the above can be solved for n, namely the number of iterations needed until a collision could occur. If the velocity the object reaches after n iterations is greater than the maximum velocity, the prediction will have to be partitioned into two segments. Namely, the segment where the velocity accelerates to the maximal velocity (this can be obtained by solving (3) for n) and the segment where the velocity stays constant.

3.3.1.2 Collision Heap

At the beginning of the simulation, the expected collision times for all object pairs are computed. The object pairs are then placed into a heap data structure [Lin92], where the object pair with the nearest expected collision time is on the top of the heap. A heap is used since most heap operations take only logarithmic time (namely, O(log p), where p is the number of collision pairs) [Sedgewick88]. At every iteration (internal time step), the expected collision time of the object pair on the top of the heap is compared to the current time. If the expected collision time is less than or equal to the current time, the object pair is removed from the heap and checked for collision. The same procedure is applied to the new top of the heap until the nearest expected collision time is greater than the current time plus the time step. If a collision did occur, the collision response is computed. Afterwards, all the object pairs removed will have a new expected collision time and can be reinserted into the heap.

Significant performance increases occur, for example, when a moving object in the environment is not near any of the complex static objects (i.e. a desk, furniture, etc.). In this case, the collision pairs between the moving object and the static objects will not be near the top of the heap and thus will incur no additional computation.

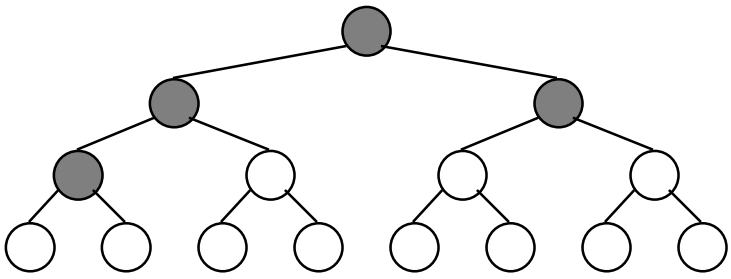


Figure 3.6: Collision Heap - example, only the shaded nodes need to be removed.

3.3.2 Static and Dynamic Objects

Since it is not known which objects will collide in an environment, it is necessary to perform collision checking on all possible pairs. This gives a maximum of $O(n^2)$ collision pairs, where n is the number of objects. Fortunately, in the environments that the VROC system tries to simulate, many of the objects are not expected to move (i.e. tables, monitors, etc.). These objects are considered *static* and no collision checking is

needed to be done between two static objects. For example, if a environment uses 100 static objects to construct a desktop and only one moving (*dynamic*) object, then only 100 object pairs are needed as opposed to the more than 10,000 pairs that would be needed otherwise.

3.3.3 Contact

Real world objects are never perfectly inelastic. In fact, most real objects will come to rest on a surface relatively quickly. The larger number of collisions that this will cause requires more computation and will reduce performance. Thus, it becomes necessary to model contact between objects in a more efficient manner.

The VROC system implements a simple contact scheme. When an object's linear and angular displacement fall below a threshold, the object is put into a contact state and does not get affected by gravity and simply rests on top of the object it came into contact with. This model works well for spherical objects; but, other object types, which come to rest with multiple contact points or perhaps with a contact plane, require additional methods.

As an additional note, friction is not implemented. The algorithms proposed by Hahn [1988] and Moore & Wilhelms [1988] include simple implementations for (sliding) friction. These were not implemented.

3.4 Control Loop

3.4.1 Host Control Loop

This section describes the PPHIGS hierarchical database created on the host and the corresponding flow of control. Section 3.5 will then describe the extensions used for the VROC parallelized implementation of the control loop.

The host workstation performs calls to the PPHIGS library to create a new database. A single structure is created. The first elements in the structure are the VROC merged world to head-mounted display matrix transformations and the view matrix transformations for each eye. As described in Section 3.1.3, PPHIGS provides a callback mechanism. A callback invocation is appended to the end of the structure. The host workstation specifies the set of VROC callback parameters for each GP. The host workstation can change the parameter values at any time.

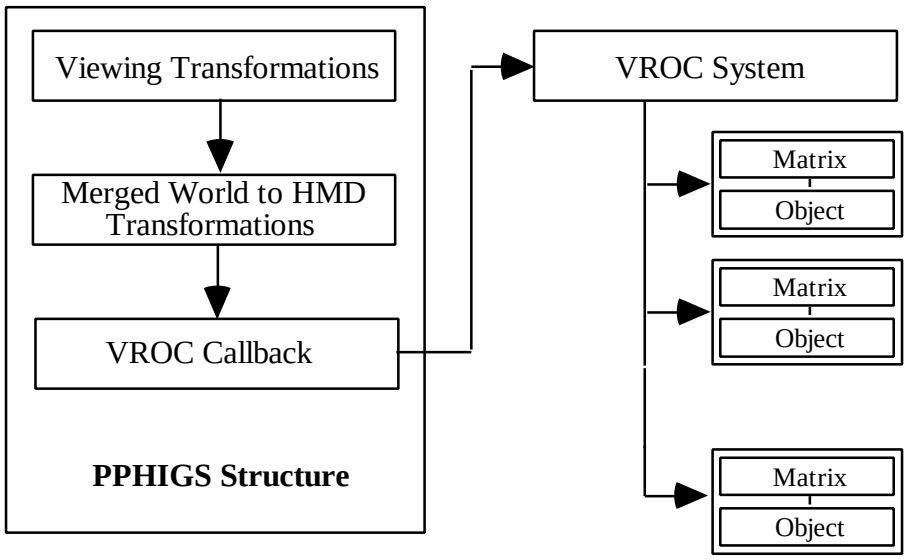


Figure 3.7: PPHIGS and VROC overall data structure

For each pair of frames (one frame for each eye), the host workstation uses the latest tracker information to compute the view matrix for each eye. During VROC's execution, the user interface edits the callback parameter values to contain the latest VROC parameter values (time step, collision distance, selected object position, etc.). Pixel-Planes 5 will then be instructed to compute the next frame for each eye.

3.4.2 Graphic Processor Control Loop

Each GP executes PPHIGS code combined with the VROC system. The GP's flow of control will eventually reach the callback invocation created by the host. The parameters for the callback contain the latest frame time step, collision distance, maximum velocities and maximum acceleration. If gravity is to be applied, its direction and magnitude are also provided. Below is an outline of the control loop implemented in the callback:

1. Perform one time initialization of the collision heap (see Section 3.3.1.2). Obtain from the callback parameters the object pairs that will be checked for collision and the necessary object definitions. The top of the collision heap will now contain the object pair that is the earliest collision candidate. It is assumed that at the very beginning of the simulation objects are not penetrating (if they are, an error message is displayed).

2. Subdivide the frame time step into steps no greater than the maximum allowable internal time step. The maximum allowable internal time step is computed based on the maximum linear velocity and the collision distance (see Section 3.2.3).

3. For each of the internal time steps, check for penetration at the end of the time step. The object pairs with expected collision time less than or equal to the current time are removed from the collision heap and checked for penetration. If penetration did occur, use binary subdivision of the time step to obtain the collision time (see Section 3.2.3). If multiple collisions occur, select the earliest one.

4. If a collision did not occur, skip to step 5, otherwise compute the collision response and update the expected collision times of the objects involved in the collision. Advance all objects to the collision time. Translate the collided objects along the collision normal so as to ensure they are not in penetration any more.

5. Advance all objects to the end of the internal time step. Goto step 3 until the frame time step has been completed.

3.5 Parallelization

3.5.1 Distribution

The VROC system lends itself well to parallelization. The collision detection scheme potentially requires a check to be performed between all possible object pairs. These checks can easily be performed in parallel. Furthermore, the collision response for simultaneous collisions between disjoint object pairs can also be computed in parallel.

The set of object pairs that have to be checked for collisions is constructed based on the static model of the real world and on the set of virtual objects that "co-exist" with the real objects. Recall from Section 3.3, that the number of object pairs is typically significantly less than h^2, where h is the number of objects in the merged virtual and real environment. The object pairs are distributed in a round-robin fashion among the multiple GPs. Each GP will construct a collision heap for the object pairs it owns. Consequently, each GP will only have to instantiate a subset of the total number of objects. An object may reside on multiple GPs, but few objects will exist on all GPs.

Additionally, each Graphic Processor draws a subset of the objects. A GP may contain objects that it does not draw, but it still needs to update those object positions over time since the object forms part of an object pair that is stored locally.

As for the distribution of collision responses, each Graphic Processor will at most compute a single collision response during each iteration (internal time step). If simultaneous collisions occur across the system, each GP will compute the collision response for its collision and broadcast the results to all GPs that have a copy of the objects involved in the collision.

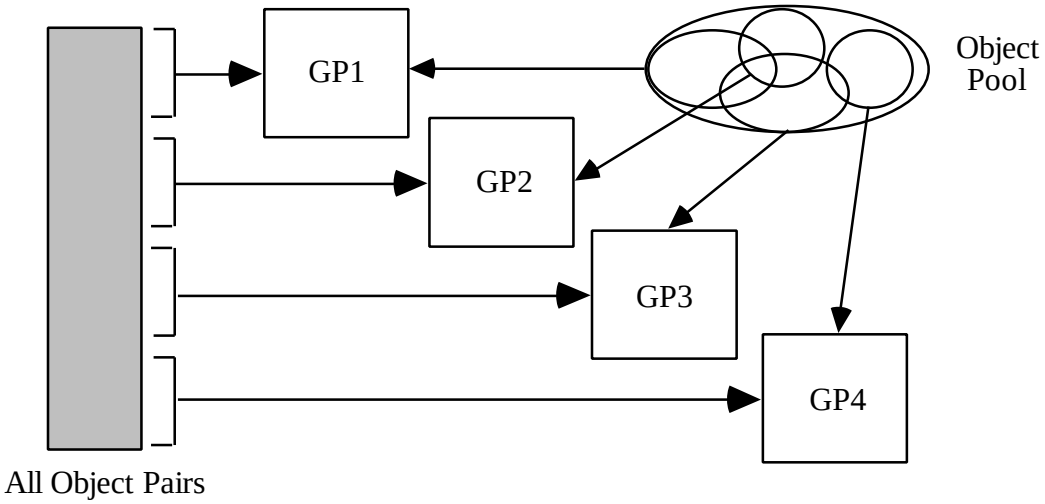


Figure 3.8: Object pair and object instantiation distribution. Each GP instantiates a subset of the objects from the object pool.

3.5.2 Message Passing

The presence of the multiple Graphic Processors, each performing a portion of VROC's computation, requires a certain amount of synchronization. Each GP will try to advance to the next frame time step. If any one GP encounters a collision, the velocities and positions of the objects involved will change. The remaining GPs need to receive the new velocities and positions. This dependency inhibits the GPs from advancing in an asynchronous manner. Therefore, messages need to be sent between the GPs in order to maintain a consistent simulated environment. Three types of messages are sent between the GPs:

- Earliest (Collision) Message
- Advance Message
- Update Message

At the beginning of a frame time step, all GPs will send an *earliest* message to the Synchronization GP (or SGP, the SGP also performs collision detection and collision response). This message indicates whether the GP encountered a collision within the next frame time step. If so, the collision time is included. If at least one of the GPs encountered a collision, the earliest collision time one (or ones) will be responded to. The SGP will send back an *advance* message to all GPs indicating until what time they should

actually advance. The GP(s) that computed the earliest time collision, will also be instructed to compute the collision response and to send the new velocities and positions to all GPs that have a copy of the objects involved in the collision. The other GPs will be informed to expect to receive a certain number of *update* messages (one for each collision). After all *update* messages have been sent and received, all GPs will try to complete the rest of the frame time step in a similar fashion.

Thus, the number of messages is independent of the number of objects. The number of messages will only increase when a collision occurs. The number of frames where a collision occurs is much smaller than the number of non-collision frames. Therefore, almost always m messages are sent to the SGP and m messages are sent from the SGP (one to each GP) per frame, where m is the number of GPs.

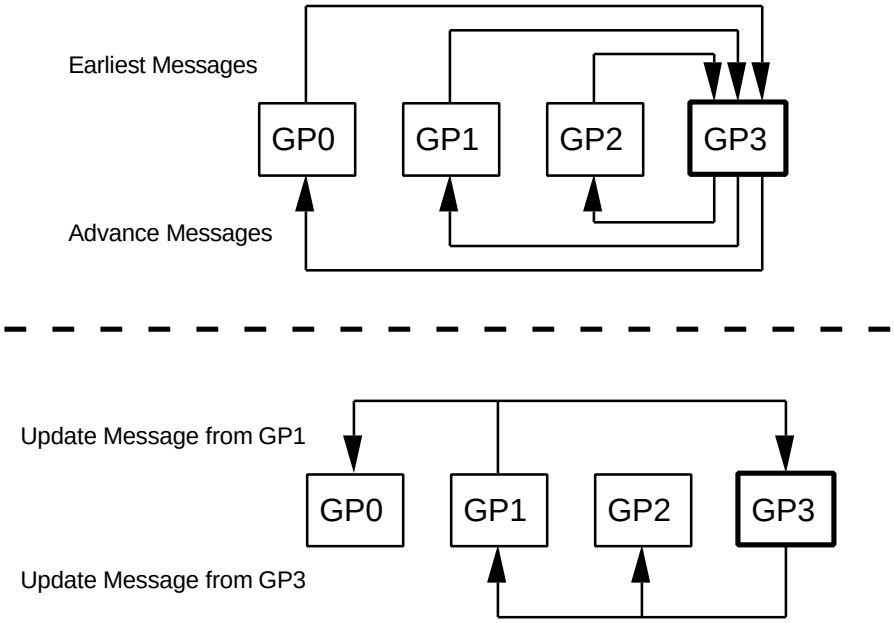


Figure 3.9: Messages in a 4 GP system, (top) messages sent at the beginning of a collision or non-collision frame, (bottom) additional messages sent during a collision frame (GP1 and GP3 computed collision responses).

Chapter 4

Sample Environments

4.1 Overview

Arbitrary environments can be designed in the VROC system. Most of the environments in this section were created using cubes and superquadrics as the basic building block. The ball (sphere) is the most intuitive object to bounce, though cubes and superquadrics are also used as moving objects in the following environments.

None of the following animations required any special coding. They are simply different input files given to the VROC system. Some of the applications require user-interaction and thus require a (see-through) head-mounted display. When the images are presented to the user in the see-through head-mounted display, the real objects are drawn in the background color (i.e. not drawn) or drawn in wireframe. For practical purposes, all figures in this section have both the virtual and real objects drawn as solid objects.

4.2 Computer Desks

This environment models a corner of the computer graphics laboratory at UNC-CH. It consists of multiple computer workstations resting on a tabletop. Various virtual objects (spheres, cubes and superquadrics) are initially above the computers. Gravity acts upon them and they fall down over the various computers.

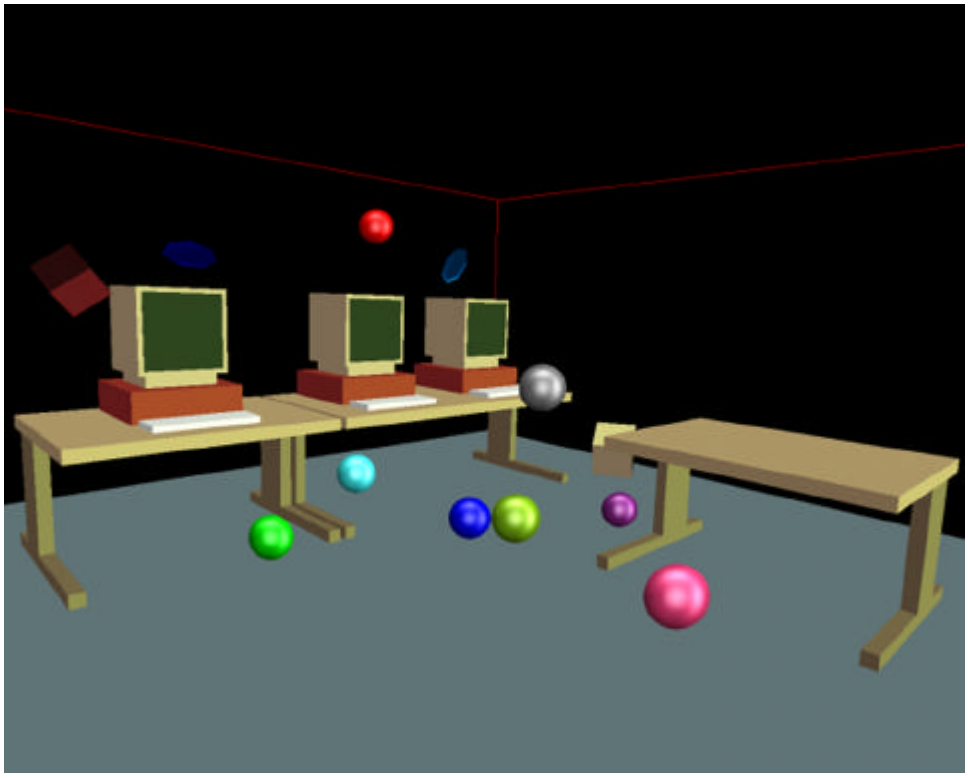


Figure 4.1: A frame of the *Computer Desks* environment.

4.3 Virtual Basketball

This environment places the user in a basketball court modeled according to official basketball court dimensions. With the help of a virtual tool panel, the user can grab a virtual basketball and throw it towards the hoop The user may alter various simulation parameters such as: elasticity, gravity, ball diameter, etc. Furthermore, a sound server was also used for this environment. Each collision generated a bounce-like sound (for obvious and unfortunate reasons, this environment has not been tested in a real basketball court).

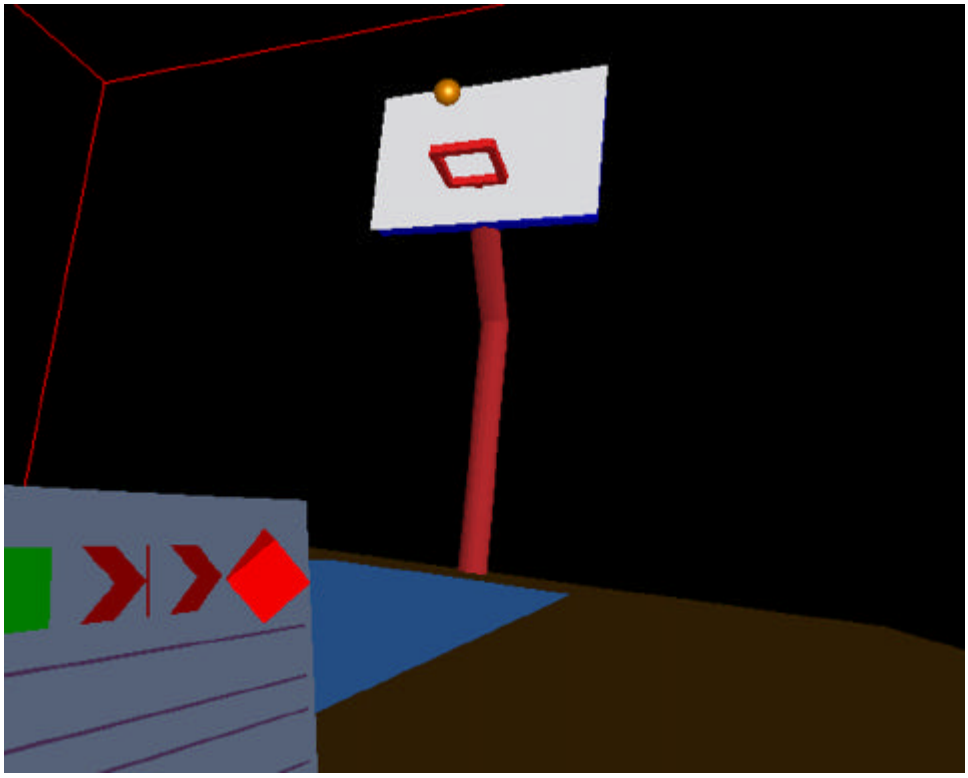
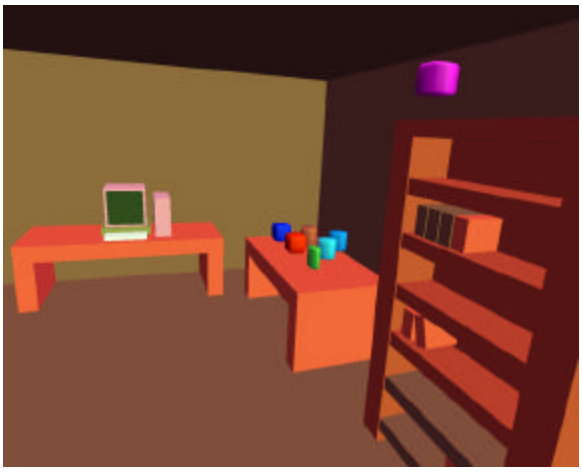


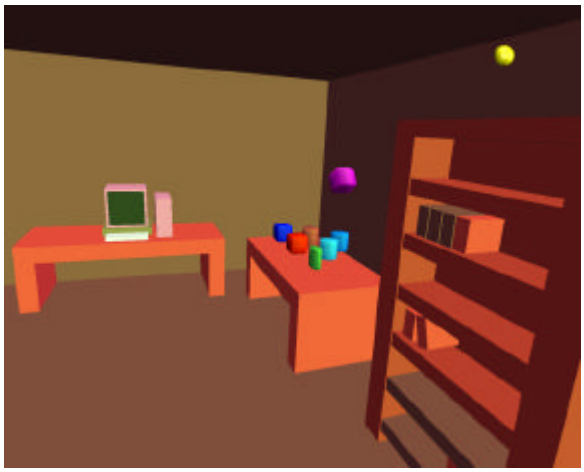
Figure 4.2: A frame of the *Virtual Basketball* environment.

4.4 Office

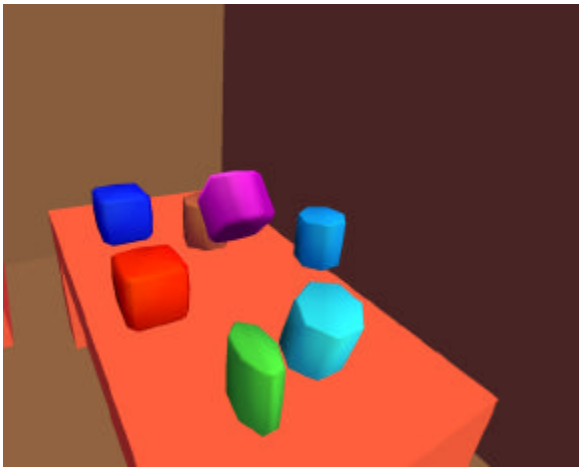
The environment represents an office with a workstation, table and bookshelf. Initially a virtual cuboid object is resting on top of the bookshelf. A small heavy (yellow) ball hits the cuboid object so that it falls on the table top knocking the tabletop objects in multiple directions.



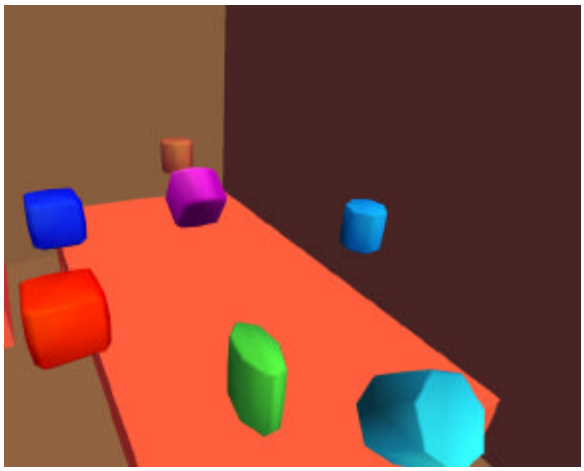
(a)



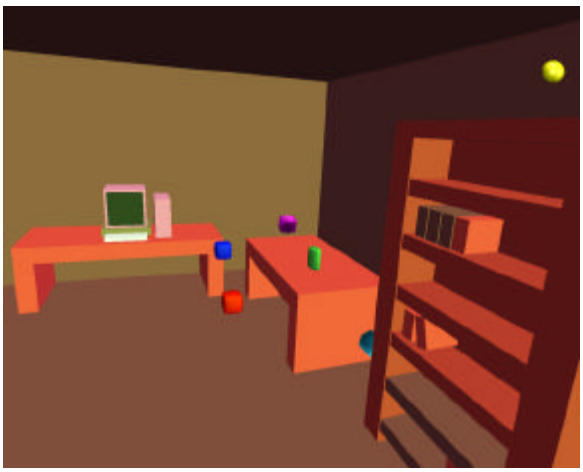
(b)



(b)



(d)

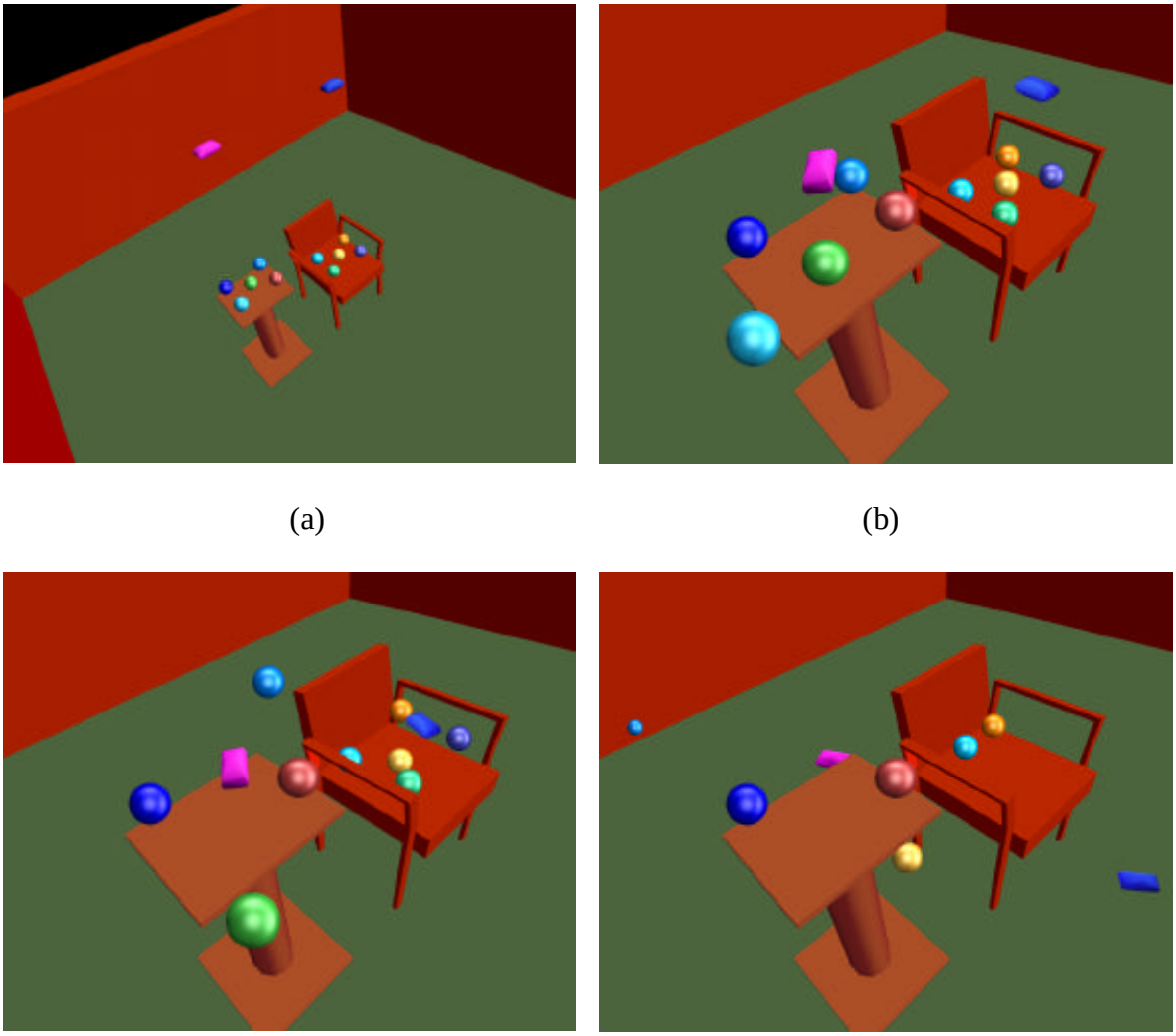


(e)

Figure 4.3: (a-e) 5 stages of *Office* .

4.5 Bounce

This environment models a chair and a pedestal which initially have a set of virtual balls on them. The user can then grab a ball and throw it at the chair or pedestal. In the below sequence, two spinning pillow-shaped objects are dropped over the chair and pedestal.



(a) (b)

(b) (d)

Figure 4.4: (a-d) 4 stages of *Bounce* .

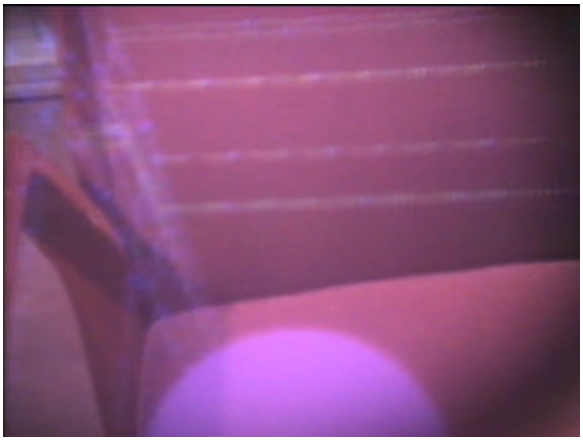
4.6 See-through Views

The following figures show a short sequence of what a user sees from within a see-through head-mounted display. The Bounce environment is used below. The user has in his hand a virtual ball which is thrown towards the balls originally at rest on the chair. Multiple collisions occur and the balls disperse in different directions bouncing off the chair bottom, back rest and arm handles.

The images were captured by placing a small camera in the location where one of the user's eye would be. The demo tape that accompanies this thesis also contains scenes from the dynamics simulator and from the combined virtual and real world.

(a)

(b)



(c)

Figure 4.5: (a-c) View from within the see-through head-mounted display of *Bounce*.

Chapter 5

System Performance

5.1 Overview

The VROC system runs at interactive to near real-time speed (between 12 and 28 Hz) depending on the environment complexity and size of the Pixel-Planes 5 configuration. This chapter describes the results of the tests over a range of environments and configurations.

The performance for a given configuration is dependent on the number of collision pairs and the frequency of collisions. The optimizations described in Section 3.3 make it possible to have moderately complex scenes and not have a quadratic increase in complexity (without the optimizations, the number of potential object pairs would increase quadratically). Nevertheless, the increase in the number of objects potentially creates more collisions and might affect performance at sporadic time intervals.

5.2 Tests

The performance measurements given below are the average frame rate (over approximately 10 seconds) for the environments described in the previous chapter. A new environment, *Simple*, is also included. This environment consists of three spheres bouncing on a tabletop. It is used to measure the maximum performance that can be achieved limited mostly by the graphics and display pipeline. Furthermore, the performance of an environment consisting of a set of randomly positioned objects is also measured (Section 5.6). This environment produces collisions distributed (approximately) uniformly over time.

The tests were run on 3 hardware configurations. The HP750 configuration is the original platform used to prototype the system, while the latter configurations use Pixel-Planes 5. The single GP version is useful to show the rendering speed of Pixel-Planes 5 versus that of the HP750. The multiple GP version shows the additional performance obtained by distributing the computations for collision detection and collision response over 25 GPs. The hardware configurations used are:

1. HP-750 TVRX-T4: The flow of control on this workstation is in essence that of a single-GP Pixel-Planes 5 system. The workstation has a single processor for the user application (PA-RISC, 64 megabytes of main memory) and four processors for the graphics pipeline processing (not available to VROC collision computations). The workstation is equipped with a standard 24-bit color display. The head-mounted display and tracker are not connected to the workstation. Hence, only one image (640x512 pixels) is generated per frame (since stereo vision is no longer required). The view direction is determined by a virtual trackball under mouse control.

2. Pixel-Planes 5, 1 GP: A minimal-size version of Pixel-Planes 5 is used. This configuration does include tracker and stereo HMD image generation.

3. Pixel-Planes 5, 25 GPs: A medium-size version of Pixel-Planes 5 is used. This is not the largest configuration, but large enough for the desired performance. The tracker used is a Polhemus Fastrak (running at an update rate of 60Hz).

The below table indicates the total number of objects, the number of object pairs actually used (remember from Section 3.3.2 that some object pairs are not considered for collisions) and the number of subframes per frame (namely, into how many internal time steps the frame time step is divided, see Section 3.2.3)

Environment	**No. Objects**	**No. Object Pairs**	**Subframes**
Simple	4	6	6
Computer Desks	43	438	2
Virtual Basketball	9	8	58
Office	43	308	12
Bounce	35	342	6

Table 5.1: Environment Characteristics

The number of subframes depends on the chosen maximum velocities, frame time step and collision distance desired. The Virtual Basketball environment requires very high velocity objects (the basketball) and a small collision distance (in order to produce accurate bouncing off the hoop). For the other environments, the number of subframes is adjusted according to their needs. Recall that the purpose of the internal time steps is to prevent objects from jumping through each other in a single frame time step. If there are multiple "thin" objects in the environment, the collision distance must be small and thus the number of internal time steps increases.

5.3 HP-750 TVRX-T4 Workstation

Environment	Frames/Sec.	Subframes/Sec.
Simple	34	204
Computer Desks	4	8
Virtual Basketball	22	1276
Office	4	48
Bounce	4	24

Table 5.2: HP-750 performance

The performance of the HP workstation is limited by the graphics pipeline. A higher frame rate could be accomplished by creating triangle meshes and a highly optimized and specialized rendering loop. Since the HP was used only as a prototype, this was not a concern.

Environment	Frames/Sec.	Subframes/Sec.
Simple	23	138
Computer Desks	1	2
Virtual Basketball	5	290
Office	0.2	2.4
Bounce	0.4	2.4

Table 5.3: HP-750 without collision heap performance

As the numbers indicate, the collision heap significantly increases performance (see Section 3.3.1.2). For the more complex environments, the performance is increased by an order of magnitude when using a collision heap.

5.4 Pixel-Planes 5, 1 GP

Environment	(Stereo)Frames/S.	Subframes/S.
Simple	28	168
Computer Desks	7	14
Virtual Basketball	21	1218
Office	7	84
Bounce	13	78

Table 5.4: Pixel-Planes 5, 1 GP performance

The above data can used be to compare the performance of a GP and Renderer(s) to the HP workstation. Recall that this configuration includes a see-through head-mounted display connection and a tracker with a 60Hz update rate.

5.5 Pixel-Planes 5, 25 GPs

Environment	Objects /GP	Pairs/GP	(Stereo)Frames/S.	Subframes/Sec
Simple	2-3	0-1	28	168
Computer Desks	26-28	17-18	18	36
Virtual Basketball	0-2	0-1	23	1334
Office	19-21	12-13	15	180
Bounce	18-21	13-14	18	108

Table 5.5: Pixel-Planes 5, 25 GPs performance

As the environments get more complex, the additional computational power of multiple GPs improves performance. Unfortunately, the presence of multiple GPs also means that messages need to be passed (see Section 3.5). But recall that the number of messages sent is (almost) totally independent of the number of objects (it only depends on the number of collisions and GPs). In conclusion, it seems that interactive to near real-time rates are maintained. The increase in performance is not linear due to the communication overhead as well as load imbalance, among other things. Some GPs might have to perform collision checks more often than others. Furthermore, a large number of non-simultaneous collisions responses will not benefit much from the presence of multiple GPs.

5.6 Additional Tests - Randomized Objects

As the number of objects increase, the number of object pairs also increases. For many environments, a fair portion of these object pairs will be discarded as static object vs. static object pairs (see Section 3.3.2). In order to measure how efficient VROC is when a large number of object pairs are present and collisions are uniformly distributed over time, an environment of multiple objects with random initial positions and velocities was generated. All objects are dynamic, thus the maximal number of object pairs need to be checked.

No. Objects	No. Object Pairs	(Stereo)Frames/S.	Subframes/Sec.
20	190	22	44
40	780	17	34
60	1770	16	32
80	3160	13	26
100	4950	12.5	25

Table 5.6: Pixel-Planes 5, 25 GPs, random objects performance

It is interesting to note the similarities in the performance of a set of 40 random objects and the Office, Bounce and Computer Desk environment. All have approximately the same number of objects but the environment with the 40 random objects has more object pairs (and the environment with 60 random objects has *significantly* more object pairs and yet approximately the same performance) than the Office, Bounce or Computer Desk environment. The apparent inefficiency of the previous three environments is probably due to the collision response computations. The objects from the random environments are approximately equally spaced and the collisions are distributed over time as opposed to the other environments where many collisions occur sporadically over a short period of time.

5.7 Performance Comparison and Analysis

The below figure overlays the performance of the multiple environments on the different configurations.

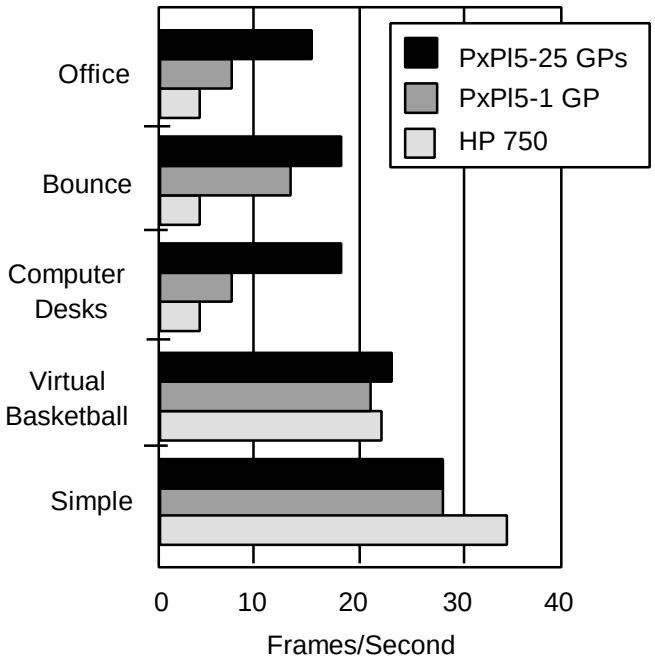


Table 5.7: Performance Comparison. Note: PxPl5 frame rates are for stereo frames and head-tracking is used to determine the view matrix.

The performance data indicates that the frame rate does not increase dramatically with the presence of additional GPs. This lack of scalability can be attributed to multiple sources, some of which are explained in the following paragraphs.

The Pixel-Planes 5 architecture has a frame rate limit of 60Hz for standard NTSC resolution images (resolution used by the head-mounted display, although the displays themselves are not necessarily of NTSC resolution). The limit is due, among other factors, to the maximum rate at which frames can be copied over the communication ring to the framebuffer. Stereo frames are generated for the head-mounted displays; thus the frame rate limit is 30 (stereo)frames per second.

In some environments the ratio of collisions to the number of objects is large. The time needed to compute the collision responses dominates the total computation time. VROC distributes among the GPs the collision detection computations and simultaneous collision responses. Specifically, if more than one pair of objects collide during the same subframe, a different GP is used to compute the collision response for each object pair. Hence as long as the number of simultaneous collisions is less than the total number of GPs, they take the same amount of time as one collision response (though

some additional time is needed for the multiple simultaneous update messages sent between GPs).

If a large number of *non*-simultaneous collision responses need to be computed over a short period of time, performance is degraded since the potential parallelism offered by the multiple GPs is not fully exploited. For example, assume that collision detection is trivial and during the next 100 subframes, 100 *non*-simultaneous collision responses need to be computed. It will take the same amount of time to compute the collision responses no matter how many GPs are present (in fact, when more GPs are present, more update messages need to be sent); a single GP can compute approximately 98 collision responses per second (while still performing collision detection and message passing). In order to efficiently handle environments where a large number of *non*-simultaneous collision responses are needed, it might be appropriate to distribute a single collision response computation among the multiple GPs rather than employing only one GP to compute a single collision response.

It is not obvious how to implement this using Pixel-Plane's 5 multi-computer architecture. The current collision response algorithm inverts a 15x15 matrix. In order to distribute such a collision response computation among the GPs, one would need to send a large number of small messages between the GPs. The collision response computation time would quickly be governed by the communication overhead between the GPs.

Exactly how much collision detection computations each GP must perform for each frame also varies during execution. Each GP maintains a collision heap. During the computations for the next frame, some GPs might have to remove from the heap and update more collision pairs than others. This is in essence a load balancing problem. The maximum through-put of the collision detection system is thus governed by the GP with the most collision pairs to process.

Chapter 6

Future Work

6.1 Overview

This chapter suggests various improvements that can be made to VROC. The order in which they are presented is approximately the order of difficulty. For some improvements, solutions are proposed, for others, pointers are given.

6.2 Object Types

The current system provides a mechanism to create environments using cubes, spheres and superquadrics as the basic building blocks. The collision detection (and collision response) can handle arbitrary convex polyhedra. In fact, all objects are represented internally as polyhedra.

A simple addition would be to allow the user to specify an input file which would define an object's polyhedral representation. Furthermore, the polyhedra need not be restricted to be convex. Lin, Manocha and Canny [1993] outlined a method for decomposing concave objects into its convex components. The collision detection method could then be applied to the individual convex components. The addition of such capabilities would expand even more the range of objects that can be used.

6.3 Scheduling Scheme

The collision system implemented takes advantage of the locality of the objects. Given a complex environment, the objects which are distant from the moving objects are not even considered for collisions (see Section 3.3.1). This can drastically reduce the computational load. The current scheduling scheme can be improved to provide a yet more accurate prediction. Currently, it does not take into consideration the direction the object is moving. For example, if an object is moving directly away from a complex part of the environment, the objects that form the complex part of the environment are still considered as possible collision candidates until the moving object is sufficiently distant that a collision with them cannot occur in any direction. For this case, the scheduling scheme could take into consideration the direction of the object since most

probably the object will not suddenly go back towards the complex environment. The only reason it would, would be if it collides with another object, in which case the complex part of the environment could be added back into the potential collision object set.

6.4 Object Pair Distribution

VROC distributes the object pairs of the simulated environment across the multiple GPs. Each GP only needs to instantiate a subset of the total number of objects. This reduces the number of update messages sent between objects when a collision occurs as well as the memory requirement per GP.

Rather than using a round-robin distribution method, a more complex method could be used. This method would distribute the object pairs among the GPs in such a fashion as to minimize the number of GPs where an object is instantiated. For example, consider a 3 GP system with an environment of 6 objects (thus a total of at most 15 object pairs):

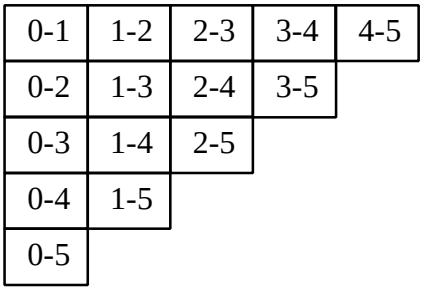


Figure 6.1: Object pairs for a 6 object environment

A round-robin distribution of the object pairs among the 3 GPs, would cause the following distribution:

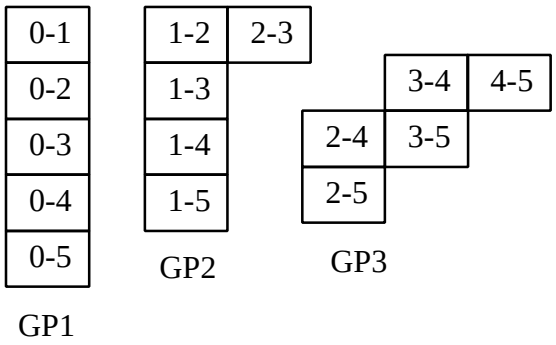


Figure 6.2: Round-robin object pair distribution

Other permutations are also possible, but in essence each GP will have to instantiate if not all the objects, at least a large number of them. An alternative would be to distribute the object pairs in an optimal fashion so as to reduce the number of objects which must be instantiated on each GP. For instance:

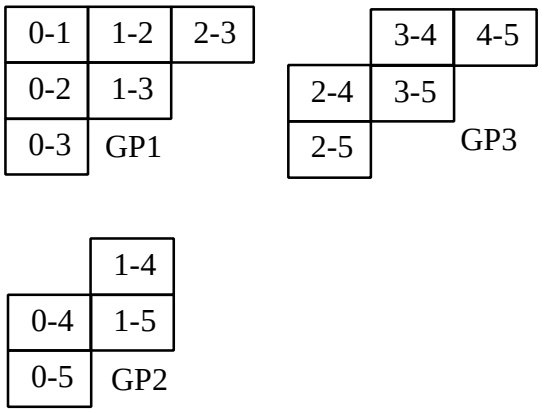


Figure 6.3: Sample improved distribution

The above distribution causes each GP to instantiate no more than 4 of the 6 objects. For a larger number of objects, the savings can be significantly greater. Additionally, this will maximize the parallel computation (since less messages need to be sent from one GP to another during a collision response) and reduce memory requirements (thus allowing for more objects in the same amount of memory).

6.5 Collision Response

Computational dynamics is a vast field. Many years of research have been put into solving the multiple problems that arise when trying to model dynamics, especially collisions. Originally, the computations were done off-line. The increase in graphics performance and computational power has allowed for some of these models to reach interactive and, in some cases, real-time rates. Below is a list of some physical phenomena (or properties) that need to be modeled to create more accurate collisions:

- Elasticity
- Friction
- Contact
- Rolling
- Constrained movements (links, etc.)
- Deformations upon impact (non-rigid objects, etc.)
- and more ...

The VROC system currently handles single point collisions of rigid objects. The model used for representing contacts is simple. Elasticity is approximated by damping the transfer of momentum.

Terzopoulos and Witkin [1988] have proposed methods for modeling flexible objects by setting up sets of dynamics equations. Baraff [1989] suggested an analytical method for handling collisions and contact forces in a system of (convex) rigid bodies. Pentland and Williams [1989] proposed a hybrid method of polynomial deformation mappings and a vibration-mode representation of object dynamics. Pentland et. al [1990] implemented ThingWorld, an interactive dynamic simulation of multibody situations. Witkin, Gleicher and Welch [1990] implemented a system for snapping together simple models (based on constraints) to create a single interactive simulation. Recent work by Bouma and Vanecek [1993] presents an efficient model for polyhedral contact in physically based simulations.

In any case, an integrated dynamics system is difficult to design, but such an addition would definitely improve the VROC system.

Section 5.7 explained some of the factors why a large number of non-simultaneous collisions does not exploit the multiple GPs well. Adding a higher granularity of distribution of collision responses might improve performance in the aforementioned situations. This would likely help to maintain a higher frame rate even when many objects suddenly collide over a short span of time.

Chapter 7

Conclusions

7.1 Overview

The VROC system integrates collision detection, collision response and head-mounted display technology to present to the user a merged world of virtual and real objects interacting. Having implementing VROC, I have learned that the following technology problems remain to be solved in order to bring such Virtual Environment applications into the practical world:

1. Obtaining a model of the real world which is of sufficient complexity for convincing interaction.

2. The visual cues presented to the user are important, but other modes of feedback (sound, force, etc.) are necessary for creating a better illusion of virtual and real objects co-existing.

3. The static calibration of real world objects and the location where the computer believes the objects lie is still a difficult problem.

4. The (dynamic) calibration problem becomes even more difficult if real world objects are allowed to move.

5. It is still a troublesome task to adequately overcome the lag introduced by the tracker and display technology.

7.2 Environment Complexity

Graphics performance is always an important issue. Fortunately, hardware is becoming fast enough to provide relatively convincing environments at real-time rates. Pixel-Planes 5 is surely capable of such a task. Once given this computational power, it becomes important to obtain an accurate model of the real world. In order to create a merged virtual and real world, the computer system must know where objects lie in the real world. As the environment complexity increases, the creation of the real world model becomes an even more time-consuming task. Creating a model of the real world does not

only include creating a correct polygonal representation, but also creating the appropriate textures and coloring for the virtual objects. The virtual objects that the user can manipulate should seem as real as their real world equivalent.

7.3 Feedback

How real the interaction of the virtual and real worlds seems to the user is not totally dependent on the visual cues from the head-mounted display. Other sensory input is also needed.

Sound feedback is important. When two real world objects collide, a specific sound is produced dependent on the objects involved. Similarly, when a virtual and real object collide, a sound should be emitted. Stereo or 3D sound would improve the realism of the merged worlds. Current audio technology is advanced enough to produce such effects reasonably well.

A more important (and difficult to implement) feedback is force feedback. The user may have a virtual object in his hand. The object might be considerably larger than his hand. It would be helpful if the user could feel when the virtual object's surface has collided with a real object. For example, an application which allows the user to place virtual furniture in an empty real room, could give the user force feedback when a virtual sofa being placed hits a wall. Another example would be the virtual basketball environment described in section 4.2. The illusion of realism would certainly be improved if the user could feel the weight and form of the virtual basketball.

Force feedback is not only useful for virtual and real object interaction but also for user and virtual object interaction. The user may wish to touch and feel the contours of a virtual object. Consider a sculpting program. Tactile feedback is essential in order to provide an effective sculpting tool.

7.4 Static Calibration

Proper calibration of the static real objects and their computer generated counterparts depends not only on a properly measured model but also on the compensation for the optical distortion generated by the see-through head-mounted display and the approximate perspective computations used for the virtual objects and computer models of real objects.

Solving the calibration problem would help to improve the apparent location of real objects from within the head-mounted display thus enabling accurate collision responses and other feedbacks (sound, force, tactile, etc.). Furthermore, virtual objects would be correctly obscured by the real objects in front of them.

7.5 Dynamic Calibration

All of the above environments have assumed a static model of the real world. This limits the range of possible environments. It is not clear how to go about removing that assumption. Information about the movement of real objects could be obtained from a tracker placed in each of the moving objects. This is implementable in the very near future, unfortunately it still does not allow total freedom of movement. Another approach would be to use imaging technology and reconstruct from a 2D camera view of the world, the 3D object's contained in it. Each 3D object would be tracked by comparing the current frame to the previous frame [Tomasi92]. In any case, this is certainly still a difficult problem.

7.6 Head-mounted Display Technology

The head-mounted displays used with the VROC system have a Polhemus 3Space Tracker (30Hz update rate) and a Polhemus 3Space Fastrak (up to 60Hz update rate). Studies done here at UNC Computer Science by Mine [1993], have measured the lag induced by these trackers to be from 10 to 30 milliseconds. If you add to this the refresh delay required for drawing a frame for each eye and the lag induced by the graphics system pipeline, you can get delays of 60 milliseconds (Pixel-Planes 5) or more (dependent on the environment complexity). This lag, though it sounds like a small amount of time, is very noticeable especially with a see-through head-mounted display. It makes the illusion of virtual objects lying on real objects must less convincing.

For example, consider a user turning his head at a rate of 200 degrees/second [Bishop84] (which is a perfectly comfortable speed; studies have shown that people regularly turn there head at speeds above 300 degrees/second; in fact, fighter pilots turn their heads at speeds in excess of 2000 degrees/second). If the user turns his head for one second and the combined tracker and graphics system introduces a lag of only 50 milliseconds, the generated image will be off by approximately 10 degrees. A typical head-mounted display has a field-of-view of 60 degrees. Thus, the image will be shifted to one side by one sixth the display resolution!

Multiple methods to compensate for this lag are being developed. A tracker with a high update rate is definitely beneficial. Reducing the graphics rendering pipeline would also minimize the lag. Unfortunately, there will always be some physical delay that cannot be easily overcome. Thus, prediction methods are also being considered as viable solutions and in fact used in some systems, such as flight simulators.

Appendix A

VROC User Interface Summary

1. Virtual Tool-Panel Interface

1.1 Overview

The VROC system has two main user interfaces: a virtual tool-panel and a command-based interface. Virtual tool-panels have been used for multiple HMD applications [Butterworth92]. From the point of view of the user in the see-through head-mounted display, the panel appears as a 45x25 centimeter flat sheet. The user can grab and position the panel wherever convenient. The panel contains multiple buttons and sliders the user may employ to control the essential features of VROC. The panel exists only as a virtual aid and does not interfere with the simulation. For most environments, the user initially uses the virtual tool panel and then places the panel out of sight (i.e. above the user's head). At any time later in the simulation, the virtual tool panel can be accessed.

1.2 Virtual Tool-Panel Controls

The virtual tool-panel has two main parts. The upper half of the tool-panel resembles a tape player used to control the simulation time:

(Stop) This button will stop the simulation time

(Start or Play) This button will start the simulation time

(Step) This button will advance one frame time step

An additional button is also present for select mode. In this mode, the user can grab any dynamic object (as opposed to static objects) in the scene by pressing the

trigger on the hand-held tracker. The object will then remain grabbed until the user releases the trigger. At this point, the object will assume the current linear and angular velocity of the hand-held tracker. This feature is used to throw virtual objects into the merged environment. When the user is not in select mode, the trigger can be used to fly [Robinett92] through the environment. Another button present on the hand-held tracker can be used to automatically re-select the previously selected object.

 (Select) Toggle for select mode

The lower half of the virtual tool-panel has two sliders. These sliders can be used to change the frame time step and the collision distance. The frame time step can range from 0.0 to 2.0 seconds (default is 0.1, a frame time step of 0 is equivalent to halting the simulation). Larger values can only be set through the command-based interface. The collision distance can range between 0.0 and 2.0 meters (default is 0.25). Again larger values can only be set through the command-based interface.

2. Command-based Interface

2.1. Object Creation Commands

2.1.1 Common

The VROC system has a command-based interface as well as a virtual tool panel. All commands can be typed in at the host workstation console prompt or can be read from a file.

The below table contains the commands needed to create a set of objects. The current system supports 3 object types: spheres, cubes and superquadrics [Barr81] [Franklin81]. The collision detection and collision response work for any convex polyhedra. For ease of use, the objects are described by generic types, rather than explicitly by a collection of triangles. Among the suggestions in future work, is to provide a simple mechanism to read arbitrary polyhedra from a separate file. Nevertheless, spheres, cubes and especially superquadrics give the user a wide range of objects. Objects must be created with a unique name. If a name is reused, the statements will be applied to the original object.

linear_vel = (x,y,z)	Set the initial linear velocity to be (x,y,z)
angular_vel = (x,y,z)	Set the initial angular velocity to be (x,y,z)
elasticity = <e>	Set the elasticity coefficient to be <e> (default is 1.0 = inelastic)
mass = <m>	Set the mass to be <m> (a negative mass for a static object)
volume = <v>	Set the volume to be <v> (default is 1.0)
translate = (x,y,z)	Set the initial translation to be (x,y,z)
rotate = (x,y,z) r	Set the initial rotation to be r radians about axis (x,y,z)
color = (r,g,b)	Set the color to be (r,g,b) (default is (255,0,0))

Table A.1: Common commands

2.1.2 Spheres

The following commands are only relevant to spheres. If applied to other object classes, they will be ignored.

radius = <r>	Set the sphere's radius to be <r> (default is 1.0)
ures = <u>	Set the sphere's u-domain resolution (default is 10)
vres = <v>	Set the sphere's v-domain resolution (default is 10)

Table A.2: Sphere commands

2.1.3 Cubes

The following command is only relevant for cubes. If applied to other object classes, it will be ignored.

size = (x,y,z)	Set the cubes (x,y,z) radii (default is (1,1,1))

Table A.3: Cube commands

2.1.4 Superquadrics

The following commands are only relevant for superquadrics. If applied to other object classes, they will be ignored.

exp1 = <e>	Set the e1 exponent of a superquadric (default is 1.0)
exp2 = <e>	Set the e2 exponent of a superquadric (default is 1.0)
a1 = <a>	Set the x-axis radius of a superquadric (default is 1.0)
a2 = <a>	Set the y-axis radius of a superquadric (default is 1.0)
a3 = <a>	Set the z-axis radius of a superquadric (default is 1.0)
ures = <u>	Set the superquadric's u-domain resolution (default is 10)
vres = <v>	Set the superquadric's v-domain resolution (default is 10)

Table A.4: Superquadric commands

2.2. Overall System Commands

The following table contains a description of the most important commands needed to run the system once a set of objects have been defined:

read <filename>	Read commands from file <filename>
display	Create a display (will get automatically called if necessary)
reset	Reset the camera transformation matrix to identity
restart	Remove all objects and restart system
start	Start time in the simulation
stop	Stop time in the simulation
step	Step only one time step
refresh	Refresh the display - do not advance in time
help	Display on-line help summary
quit	Quit the system
time_step = <t>	Set the time step to be <t> (i.e. 0.1)
collision_dist = <d>	Set the collision distance to be <d> (i.e. 0.25)
internal_step = <i>	Override the internal time step and force it to be <i>
max_linear = <v>	Set the maximum linear velocity to be <v> (default is 8.0)
max_angular = <v>	Set the maximum angular velocity to be <v> (default is 4.0)
force = (x,y,z) m	Set global force to all objects in direction (x,y,z), magnitude m
gravity	Set global force to be gravity (i.e. (0,0,-1) 9.8)
collision_sound = <s>	Using the sound server, generate sound <s> on collisions
collision_vol = <v>	Set the volume of the object collision sound

Table A.5: System commands

References

[Bajura92] Bajura M., Fuchs H., Ohbuchi R., "Merging Virtual Objects with the Real World: Seeing Ultrasound Imagery within the Patient", *Computer Graphics*. vol. 26, no. 2, July 1992.

[Baraff89] Baraff D., "Analytical Methods for Dynamic Simulation of Non-penetrating Rigid Bodies", *Computer Graphics (Proc. SIGGRAPH)*, vol. 23, no. 3, pp. 223-232, July 1990.

[Baraff90] Baraff D., "Curved Surfaces and Coherence for Non-penetrating Rigid Body Simulation", *Computer Graphics (Proc. SIGGRAPH)*, vol. 24, no. 4, pp. 19-28, August 1990.

[Baraff92] Baraff D., "Rigid Body Dynamics", *Computer Graphics Course Notes: An Introduction to Physically Based Modeling (Proc. SIGGRAPH)*, pp. H3-H29, 1992.

[Baraff93] Baraff D., "Issues in Computing Contact Forces for Non-Penetrating Rigid Bodies", *Algorithmica*, vol. 10, no. 2/3/4, pp. 292-352, August-October 1993.

[Barr81] Barr H., "Superquadrics and Angle-Preserving Transformations", *IEEE Computer Graphics and Applications*, vol 1., no. 1, pp. 11-23, January 1981.

[Bishop84] Bishop G., "Self-Tracker: A smart Optical Sensor on Silicon", Ph.D. Dissertation, University of North Carolina at Chapel Hill, TR. 84-002, 1984.

[Butterworth92] Butterworth J., Davidson A., Hench S., Olano M., "3DM: A Three Dimensional Display Using a Head-Mounted Display", *Proceedings of 1992 Symposium of Interactive 3D Graphics*, Cambridge, MA, March 29-April 1, 1992.

[Cameron90] Cameron S., "Collision Detection by Four-Dimensional Intersection Testing", *IEEE Transactions on Robotics and Automation*, vol. 6, no. 3, pp. 291-302, June 1990.

[Canny86] Canny J., "Collision Detection for Moving Polyhedra", *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. PAMI-8, no. 2, pp. 200-209, March 1986.

[Ellsworth90] Ellsworth D., Good H., Pixel-Planes 5 Hierarchical Interactive Graphics System (PPHIGS) - Programmer's Manual, University of North Carolina at Chapel Hill, NC, November 1990.

[Franklin81] Franklin W. R., Barr H., "Faster Calculation of Superquadric Shapes", *IEEE Computer Graphics and Applications*, vol 1., no. 1, pp. 41-47, July 1981.

[Fuchs89] Fuchs H., Poulton J., Eyles J., Greer T., Goldfeather J., Ellsworth D., Molnar S., Turk G., Tebbs B., Israel L., "Pixel-Planes 5: A Heterogeneous Multiprocessor Graphics System Using Processor-Enhanced Memories", *Computer Graphics (Proc. SIGGRAPH)*, vol 23, no. 3, pp. 79-88, July 1989.

[Glassner90] Glassner A., *Graphics Gems*, Academic Press, Inc., 1990.

[Goldstein50] Goldstein H., *Classical Mechanics*, Addison Wesley, Reading, MA, 1950.

[Hahn88] Hahn J., "Realistic Animation of Rigid Bodies", *Computer Graphics (Proc. SIGGRAPH)*, vol. 22. no. 4, pp. 299-308, August 1988.

[Herzen90] Herzen B., Barr A., Zatz H., "Geometric Collisions for Time-Dependent Parametric Surfaces", *Computer Graphics (Proc. SIGGRAPH)*, vol 24., no. 4, pp. 39-48, August 1990.

[Kirk92] Kirk D., *Graphic Gems III*, Academic Press, Inc., 1992.

[Li90] Li Z., Canny J., "Motion of Two Rigid Bodies with Rolling Constraint", *IEEE Transactions on Robotics and Automation*, vol. 6, no. 1, pp. 62-72, February 1990.

[Lin91] Lin M., Canny J., "A fast algorithm for incremental distance calculations", *IEEE International Conference Robotics and Automation*, pp. 1008-1014, 1991.

[Lin92] Lin M., Canny J., "Efficient Collision Detection for Animation", *Third Eurographics Workshop*, Cambridge, England, September 1992.

[Lin93] Lin M., Manocha D., Canny J., "Fast Collision Detection between Geometric Models", University of North Carolina at Chapel Hill, TR 93-004, January 1993.

[Mine93] Mine M., "Characterization of End-to-End Delays in Head-Mounted Displays", University of North Carolina at Chapel Hill, TR 93-001, 1993.

[Moore88] Moore M., Wilhelms J., "Collision Detection and Response for Computer Animation", *Computer Graphics (Proc. SIGGRAPH)*, vol. 22, no. 4, pp. 289-297, August 1988.

[Pentland89] Pentland A., Williams J., "Good Vibrations: Modal Dynamics for Graphics and Animations", *Computer Graphics (Proc. SIGGRAPH)*, vol. 23, no. 3., pp. 215-222, July 1991.

[Pentland90] Pentland A., Essa I., Friedmann M., Horowitz B., Sclaroff S., "The ThingWorld Modeling System: Virtual Sculpting By Modal Forces", *Proceedings of 1992 Symposium of Interactive 3D Graphics*, vol. 24, no. 2, pp. 143-144, Snowbird, Utah, March 25-28, 1990.

[Pentland90b] Pentland A., "Computational Complexity Versus Simulated Environments", *Computer Graphics*, vol. 24, no. 2, pp. 185-192, 1990.

[Pimentel91] Pimentel K., Teixeira K., *Virtual Reality: Through the new looking glass*, Windcrest McGraw Hill, 1992.

[Robinett92] Robinett W., Holloway R., "Implementation of Flying, Scaling and Grabbing in Virtual Worlds", *Proceedings of 1992 Symposium of Interactive 3D Graphics*, Cambridge, MA, March 29-April 1, 1992.

[Sclaroff91] Sclaroff S., Pentland A., "Generalized Implicit Functions for Computer Graphics", *Computer Graphics (Proc. SIGGRAPH)*, vol. 25, no. 4., pp. 247-250, July 1991.

[Sedgewick88] Sedgewick R., *Algorithms*, Addison-Wesley Publishing Company, Inc., Princeton University, 1988.

[Terzopoulos88] Terzopoulos D., Witkin A., "Physically Based Models with Rigid and Deformable Components", *IEEE Computer Graphics and Applications*, November 1988.

[Terzopoulos91] Terzopoulos D., Metaxas D., "Dynamic 3D Models with Local and Global Deformations: Deformable Superquadrics", *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 13, no. 7, pp. 703-714, July 1991.

[Tomasi92] Tomasi C., Kanade T., "Shape and Motion from Image Streams under Orthography: a Factorization Method", Intl. Journal of Computer Graphics, Vol 9, no. 2, pp. 137-154, 1992.

[van Dam88] van Dam A., Chairman, PHIGS+ Committee, "PHIGS+ Functional Description, Revision 3.0", *Computer Graphics*, vol. 22, no. 3, pp. 125-218, July 1988.

[Wilhelms88] Wilhelms J., Moore M., Skinner R., "Computer Animation Based on Dynamic Simulation", *Proc. International Conference on Computer Graphics*, pp. 85-95, 1988.

[Witkin90] Witkin A., Gleicher M., Welch W., "Interactive Dynamics", *Proc. 1990 Symposium on Interactive 3D Graphics*, 1990.