building games in VRML

peter gerstmann
may 2000

acknowledgements

I would expressly like to thank the following people for their contributed time, patience, expertise, and support. It is only with their help that this paper could have been completed.

Matt Lewis
Lawson Wade
Steve May
Neal McDonald
Wayne Carlson
Susan Roth
Dale Gerstmann
Lucia Gross
Derek Gerstmann

I am also endebted to the pioneers of VRML community, for providing an open format for Web3D, inspiration, software, and example code. Many thanks to:

The old Cosmo team at SGI
Parallelgraphics
Blaxxun
Shout Interactive
Pioneer Joel
Chris Marrin
Don Brutzman
David Frerichs
Braden McDaniel
Vladimir Bulatov
Michael Wagner
Stephen White
Roland Smeenk
Tom Kaye

abstract

This research focuses on the process of creating 3D games using VRML, the Virtual Reality Modeling Language. It presents the Web3D community with a library of components and utilities to be used together to simplify VRML game production and proposes a tool to facilitate working with VRML components in general. Design of the components is inspired by encapsulated model theory and guided by analysis of popular games and the VRML production process. Illustration of their use is provided in the form of working examples. Source code for the examples is provided.

background

the 3D graphics boom

In recent years, the combined availability of faster, cheaper computer processors, less expensive memory, and more powerful mass market graphics accelerator cards has resulted in real-time 3D graphics usage becoming widespread among the casual computer user. 3D computer games have flooded the market, becoming the norm rather than the exception. With more and more complex 3D animation techniques appearing in movies and on TV, general public exposure to 3D graphics has risen sharply.

Real-time 3D on the web became a reality in 1995 with the VRML 1.0 specification and several VRML-capable browsers. Since then, VRML 2.0 has become an international standard (VRML97). VRML2000 is being standardized as X3D. Java3D and a host of other Web3D technologies are now available. VRML97 is currently the most widely supported standard open format for complex real-time interactive 3D scenes on the web.

[01], [61]

what is VRML?

VRML is a scene description language, similar to HTML being a document description language. A scene is composed of a list of objects, called nodes. Nodes describe things such as shapes, colors, lights, viewpoints, and transformations. Nodes are grouped and nested to form a hierarchical structure that defines the scene of interest. The following is a simple VRML file that describes a cube. (See figure 1)

```
#VRML V2.0 utf8

Background { skyColor [ 1 1 .9 ] }
Shape {
   appearance Appearance { material Material {} }
   geometry Box { size 1 1 1 }
}
```

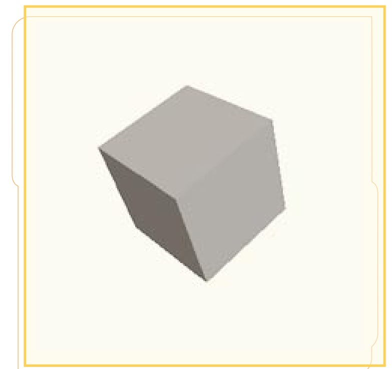


fig01: a simple VRML file

VRML is an excellent tool for 3D visualization, enabling not only static model representation with full texturing / coloring capabilities, but also animation and interaction. Furthermore, VRML scenes can be built from other VRML scenes to achieve arbitrary levels of complexity.

motivation

Robust technology for the delivery of interactive 3D coupled with inexpensive computing power positions the common computer user to take advantage of 3D over the web.

Given the popularity of off-line 3D games then, one might expect to see an abundance of on-line 3D games. This is not the case however. There are presently only a handful of poorly publicized 3D games on the web.

[02], [03], [04], [05]

Before online 3D games can begin to compare to their offline relatives, developers will need access to tools as powerful as those they've been using for their offline games. Libraries of reusable code components need to be built to support common functionality and reduce development effort. Object-oriented programming design methodolgies are extremely applicable to this type of situation.

thesis

Before online 3D games can become comparable their offline counterparts, appropriate resources need to be made available to the developer. This research chooses VRML as an open, standardized format for delivering online 3D content. In this context, *encapsulated model theory* can be applied to the game design process to develop modular, reusable, shareable libraries of VRML components for game production. However, tools that allow the developer to effectively utilize these libraries will also need to be developed.

applying encapsulated model theory to vrml

what are encapsulated models?

[12] May defines encapsulated models (emodels) as:

> *...an animated object containing an integrated set of dynamic attributes – e.g. shape, motion, materials (surface properties), light sources, cameras, user interfaces, sound – represented by a procedural data format (i.e. a program written in a procedural animation language).*

Real-world models almost always employ forms of encapsulation to provide the user with a simple interface to control a complex object. Consider an electrical toy robot. Any child can easily get the robot to walk, blink lights, speak, and fire its laser gun, simply by flipping a power switch from off to on. It's not important that the child know anything about the complex circuitry that exists inside the robot's body.

Virtual-world models can similarly employ encapsulation. As an example, imagine the case of a 3D artist wanting to animate a modeled character's face. To express an emotion such as happiness or sadness will require the manipulation of several, if not hundreds, of points on the surface of the model. These same points will need to be manipulated every time the character's expression changes. Suppose instead of maneuvering each individual point, the animator could instead adjust a single simple value that would [13] indicate the degree of happiness; maybe 0 for frowning, .5 for neutral, and 1 for a big smile, with smooth transitions for all the values in between. Obviously, the process of completing the animation will be much shorter and more efficient. Even if the artist knows all there is to know about character animation, it will still be beneficial to encode the expressions once, rather than having to manipulate so many points repeatedly. Additionally, and importantly, the model can then be reused by someone not necessarily as skilled in character animation, because one would only need manipulate the single happiness value. This leads into the concept of emodels as an improved data format for model distribution. Imagine the decrease in production time if off-the-shelf models from [14] commercial companies came not as static collections of points, but "with predefined movable parts, built-in animation controls, built-in sound effects, and parameters to vary the shape, style, or functionality of the model."

[15] The virtues of emodels are twofold: increased complexity and sophistication of objects, and decreased cost in time and effort to use them.

These benefits are functions of the properties of emodels, namely, [16]

procedural specification - a program defines the model
parameterization - aspects of the model can be tied to changeable values
replication - multiple copies of the model can be made
precision - model parts can interact in mathematically precise ways
continuous representation - equations define shapes, rather than point lists
compression - the data format is space-efficient
composition - complex models can be built from simpler models
simulation - the ability to implement particle systems, behavior simulation, etc.
lighting - lights, surface materials and atmospheric effects described in the model
sound effects - sounds described in the model

Emodels are compact, self-contained, reusable, and customizable. Modularity of this sort is ideal for any constructive process, be it animation, programming, or game production.

If emodels could be created with VRML as the representation language, 3D game designers could take advantage of all the properties of emodels listed above. In a significant step towards the goal of simplifying game design, emodel libraries could be established to promote the sharing (or selling) of complex models. These models could be props, characters, behaviors, tools, etc., ready for use in a game.

VRML as an emodel representation language

The VRML97 specification suggests support for scripting languages. Language choice is left to the implementers of the browser software, but is typically ECMAscript and / or Java. When implemented with a scripting language, VRML can be used to create models that exhibit all of the properties of emodels listed above. [17], [63]

Encapsulation in VRML is achieved through the PROTO and EXTERNPROTO structures. A PROTO takes an arbitrary collection of nodes and bundles them into a single node with a single interface for sending and receiving data. Recall that nodes are the atomic unit in VRML and can represent shapes, colors, lights, textures, sounds, motions, transformations, etc. The PROTO structure is simply a wrapper node that allows other nodes to hide inside of it. This can be thought of as analogous to a stereo component supplying input and ouput jacks for audio, while hiding the circuitry of its audio processing electronics inside a box. [18], [64]

The EXTERNPROTO structure is a PROTO that resides in a separate (external) file. This allows complete encapsulation and extensive reuse. A single EXTERNPROTO file can be referenced by any number of different VRML models, simply by including its parameter interface and a link to its implementation.

VRML models can therefore be emodels when they are defined as EXTERNPROTOs. Their parameter interface allows the user to customize the model to meet specific needs. VRML emodels can themselves be composed of other emodels. This recursive definition allows progressively more complex objects to be built from basic components.

To illustrate the applicability of VRML to emodel theory, an example VRML emodel has been created: *RocketClock*, which is similar to *AlarmClock* presented by May. [20] The model exhibits several high level controls such as clock color, setting of the time and alarm, alarm volume, and exaggeration of ringing motion. Furthermore, *RocketClock* makes use of another emodel, *ClockMechanism*, to drive the rotations of the hands around the face. The *ClockMechanism* behavior emodel could easily be reused in another clock model, which might have completely different geometry.

The *RocketClock* example is presented on page four, and its implementation is given in Appendix C.

Rocket Clock

fig02: RocketClock

fig03: RocketClock alarm

RocketClock is a whimsical night-stand clock, something which might be found in a child's bedroom. It displays the current time with hour, minute, and second hands that rotate around the cockpit portal (figure 2). If the alarm is set and the current time matches the alarm time, then the rocket will emit a light like an engine firing, play a lift-off sound, and vibrate around in little circles to simulate the turbulence of launching (figure 3). The rocket's color, shininess, light color, sound, sound volume, turbulence intensity, time and alarm can all be set by the user.

Time is kept inside the rocket clock via the *ClockMechanismPROTO*, a behavior emodel that simulates a clock engine. Independent of geometry, it generates the rotation angles for the hour, minute and second hands of an arbitrary face. It has parameters for setting the hour, minute and second of both the starting time and alarm. Additionally, the 'auto-clock' parameter is a switch that, if on, will ignore any starting time input values, and set the clock to the time on the user's computer.

emodel libraries

When considering all the benefits of using emodels, one would expect to see vast libraries of them available to developers.

[21], [22], [23], [24], [79]

Collections of VRML emodels do exist. Most of them, however, benefit game developers only indirectly at best, by implementing buttons, textures, text displays, or geometry generators. VRML emodel use and development is hampered by the fact that almost no VRML creation software supports the EXTERNPROTO structure.

It is the goal of this research to contribute a starting library of game related VRML emodels as well as a prototype development environment for working visually with EXTERNPROTO structures, in the hope that commercial VRML software companies will be inspired to advance their software in this direction.

The process of designing useful components for game production must first include an analysis of games in general, as well as an analysis of the typical game production pipeline. Common desirable behaviors in games and weaknesses in available production tools can then be identified and addressed.

A library of useful components would be further enhanced by a tool to allow their inclusion and manipulation in VRML files through a visual interface. Such a tool would empower novice and advanced VRML developers alike, affording them easy access to the benefits of visual building with component libraries.

Game analysis begins on page five, followed by a discussion of the game production pipeline on page eight. A prototype tool is described beginning on page ten.

popular games

To identify emodels that would be useful to game designers, it is necessary to examine the most popular games and look for common aspects that could be encapsulated into emodels. Three lists of the 'Greatest Games of All Time' were compared from three popular video game magazines. The fifty highest ranked games from each list can be found in appendix B.

[29], [30], [31]

categories

Each of the top ranked games fit into one of the following general categories:

category:	emphasis:
Action:	aiming and maneuvering; chase / flee games often occuring in a large environment **Ex:** Asteroids, Missile Command, Quake, Robotron
Arcade:	agile character control and careful timing **Ex:** CastleVania, Crash Bandicoot, Sonic
Fighting:	quick reaction times and move sequence planning, often occuring in a restricted environment **Ex:** Soul Blade, Street Fighter, Tekken
Graphic Adventure:	exploration and puzzle solving, usually in large, richly graphical environments **Ex:** King's Quest, Myst
Puzzle:	quick solutions to graphical puzzles **Ex:** Attax, Intelligent Qube, Tetris
Real-Time Strategy:	resource management and positional strategy, often occurring in large, unrealistic environments **Ex:** Command & Conquer, Myth
Resource Management:	commodity allocation, usually in a slower paced, turn-based setting **Ex:** Civilization, Populous, SimCity
Role Playing Game:	character evolution through extended game play, usually involving very large environments, complex storylines, and multiple protagonists **Ex:** Final Fantasy, Phantasy Star
Simulation:	realistic physical simulation of subject, often occurring in large, realistic environments **Ex:** Jane's Combat Sims, Red Baron, Formula I
Sports:	skills related to the subject sport **Ex:** NFL 2000, PunchOut, Wayne Gretzky Hockey
Text Adventure:	narrative and puzzle solving, usually involving detailed environments and elaborate storylines **Ex:** Zork I, II, III

common functions

Focusing on the categories of games that are typically played in realtime 3D, the next step is to identify functionality that has to be implemented every time a game in one of these categories is produced. Such common functions will be ideal candidates for emodels. Listed below are the focus categories and common characteristic functions.

Action:	automatic character guiding and orienting
Arcade:	navigation control, dynamic geometry fracturing
Fighting:	intelligent camera positioning / aiming
Puzzle:	opponent AI
Real Time Strategy:	terrain generation, pathfinding / obstacle avoidance
Simulation:	physics engine, HUD display, visibility / containment detection
Sports:	collision detection, human animation

Note: Most of the functions are re-used across multiple categories, but are only listed above once, in the category deemed most characteristic of that function.

interface considerations

In order for the user to interact with the game environment, there has to be a user-game interface. User input in VRML is generally mouse input, and VRML provides nodes to sense mouse actions. However, examples of using Java to sense keyboard input have been implemented, and some browsers support keyboard input extensions to VRML. Considering mouse and keyboard input, there are several commonly found user interface behaviors: [32], [77], [78]

button	slider / lever	knob
trackball	mousepad	pop-up menu

Of these, slider, knob, lever, trackball, and mousepad are fairly easy to implement in VRML, and emodels already exist for many of them. However, a behavior to handle the general case of switching geometry based on mouse input would enable the creation of buttons and pop up menus. Such a behavior is often referred to as a 'rollover'. [21], [79]

appropriate emodels

Looking at the compiled lists of common functions, there are a few that are beyond the scope of this research. A robust real-time physics engine is a large project unto itself. Accurate real-time 3d collision detection and intelligent automatic camera positioning are also involved projects. The following list describes the emodels implemented for this research. They are presented in three categories: behaviors, sensors, and utilities. [35]

behaviors:

AvoidObstacles works with a subject and a group of obstacles to guide the subject towards a goal while staying away from the perimeter of the obstacles to avoid.

Guide also directs a character, but does so in response to a single goal. The subject is guided towards (chasing), or away from (fleeing) the goal.

ExplodingPolySet disperses the polygons of an object, much like a grenade turns to shrapnel. It was written by Bulatov, and is included here due to its strong relevence to games. [65]

GlueToView allows geometry to appear fixed to the screen, even as the viewpoint is changing. This is useful for HUDs, or Heads Up Displays, often employed in games to communicate location, progress, score, etc.

LookAt generates rotation values to orient a subject to face a target object.

ClockMechanism was written for the *RocketClock* example introduced earlier. It produces rotation values to drive the hands of a clock face.

sensors:
BoundingSphereCollisionSensor detects collisions between a subject and an array of obstacles, returning the indices of the obstacles collided with.

RolloverSensor switches geometry based on four mouse triggered states:

mouseOff:	the initial state
mouseOver:	when the user positions the mouse pointer over the sensor geometry
mouseDown:	when mouseOver is true and the left mouse button is pressed
mouseUp:	when mouseOver is true and the left mouse button is released

utiltities:
To make the implementation of the behavior and sensor emodels possible and / or easier, several utility emodels were developed.

GetBoundingBox returns the dimensions of the smallest box that will fit around all of its child object's geometry. The dimensions are given as two points, which represent opposite diagonal corners of the bounding box.

GetBoundingSphere returns the center and radius of the smallest sphere that will fit around all of its child object's geometry.

Noise is a noise generator as described and implemented by Peachey. Noise is a common concept in image and signal processing theory (i.e. static), and was introduced to the computer graphics community by Perlin in his 1985 Siggraph paper. Essentially, a noise generator produces output values that look random, but are repeatable when the same input values are supplied. This makes noise controllable. Uses for noise typically involve introducing random variations into regular patterns; to turn stripes into marble, a flat plane into mountains, etc. For the *RocketClock example introduced earlier*, noise is applied to the horizontal position of the rocket, to simulate launch vibrations.

[75]

[76]

Output prints a text stream to a new window, providing control over the window parameters and text's MIME type.

WaveInterpolator returns the sine or cosine of a given input value.

Detailed descriptions of the emodels are in Appendix B, listed alphabetically. Implementation code is given in Appendix C.

Game production is a pipeline from original concept to playable product. To decide what will be most useful to the VRML game designer, the process as a whole needs to be broken down into discrete steps, toolsets need to be identified and evaluated for each step, and emodels need to be designed to complement the weaker toolsets.

the game development pipeline

This research approaches VRML game design as a process of six major steps:

storyboard	design on paper the key aspects of the gameplay, rules, and characters
encapsulate	create, collect, and encapsulate attributes (shape, motion, materials, lights, camera views, sound) to form emodels for use in the game
compose	arrange multiple emodels together to compose game environment
program	code the neccessary scripts to manage interaction of emodels and user input
test	play test the game, check the behaviors, find and fix bugs
publish	collect separate files into single files, validate and optimize VRML code for download and processing efficiency

leaks in the pipe

Each of the steps in the game design process requires specific, specialized tools. Areas with solid support and accessibility include:

	storyboard	pen and paper are certainly very accessible and cheap
	encapsulate	
[09], [10], [11], [38], [39], [40], [41], [42] [43], [44]	shapes	There has been an explosion of 3D modeling tools in recent years, covering the full range of functionality and price. Many will export VRML directly, and for those that don't, there are utilities that will convert from almost any file format to VRML.
[66], [67], [68], [69]	materials	2D graphics packages have been around for a long time, and there are many options to choose from for texture image creation.
[70], [71], [80]	sound	Sound creation tools are also well established and available.
[07], [08], [81]	**compose**	Several capable VRML specific creation / composition tools are available for reasonable prices ($100 or less). None, however are directly targeted for game design, and very few support the EXTERNPROTO structure.
[72], [73], [74]	**program**	Robust text editors for VRML coding are readily available.
[45], [46], [47], [48]	**test**	Free validation software for VRML exists, and some of the major browsers have error feedback built in
[46], [47], [72], [07]	**publish**	Stand alone optimization tools are available, and some of the composition tools and text editors have built in optimization support.

The areas with the weakest support or least accessible tools are:

encapsulate	
textures	Most VRML building tools support texture application in some form, but none support high level texturing procedures well, such as interactive 3D painting, or texture 'unwrapping'~ creating an unfolded version of the geometry as a drawing template.
lights	Lights are naturally invisible in VRML; the only evidence of their existence is the light they emit. Many VRML creation tools support light creation, but even their iconic representations usually don't illustrate all the physical properties of the light, such as direction, fall-off area, attenuation, and beam width. Regardless, a visual representation in VRML would be useful.
sound	Sounds are also naturally invisible in VRML, and few, if any, VRML building tools show the actual geometric volumes of the sound (sounds in VRML emit noise in 3D space in the shape of a user-defined ellipsoid).

encapsulation	There are currently no visual tools that allow use or creation of new nodes using the EXTERNPROTO construct, which discourages the growth and utilization of libraries of emodels.
compose	The bulk of most VRML creation software is support for composing VRML scenes, but a few simple manipulation tools that worked directly in VRML, without the need for any external software, would be useful, and more accessible.
program	A couple VRML building tools provide support for wiring the basic VRML nodes together (sending the output of one node to the input of another), but none allow for custom nodes to be added and hooked up. As mentioned on page four, there is also a lack of predefined game related behaviors to augment the basic VRML nodes.

[06], [82]

appropriate emodels

Based on analysis of the separate steps of the game development pipeline, the following emodels were designed to provide support for the weaker toolset areas.

encapsulate:
Visual support for lights is provided with *VisibleDirectionalLight*, *VisiblePointLight*, and *VisibleSpotLight*. These emodels graphically reflect the light parameters such as color, direction, radius or falloff, etc., which would otherwise be visible only second-hand, by introducing an object in the scene to be affected by the light.

Visual support for sound nodes is provided with *VisualSound*, which shows the user a physical representation of the sound volume in 3d space: its origin, minimum and maximum ranges.

Advanced texturing support is a bit too involved for the scope of this research, and is left as a future project possibility.

compose:
Visual support for transformations is provided with *VisibleTranslate*, *VisibleRotate*, and *VisibleScale*. These interactive emodels provide handles to translate, rotate, or scale their child objects directly in the VRML environment. The numeric values of the transformations are printed to the browser's status bar for reference.

Access to orientation / position information for viewpoint, transform, sound and light nodes is provided with the *Info* emodel. By using the standard VRML navigation mechanisms, the user can position their view to be where they would like a sound, light, viewpoint, etc. located, and click on any of the emodel's icons to have the appropriate node's information printed to the browser's VRML console for easy cut and paste into their code.

The *AxisJack* emodel is a very simple model, simply showing the location of the origin and coordinate axes. Rolling over any of the axes with the mouse prints the axis's label to the browser's status bar. This is often useful for composing scenes in the void of empty space, since VRML has no visible axis reference of its own.

program:
Designing emodels for the programming phase of game creation was covered in the previous section: *common game behaviors*.

Detailed descriptions of these emodels can be found in Appendix B. Their implementations can be found in Appendix C.

vrml emodel development environment

To fully exploit the power of the emodel concept and the flexibility of reusable component libraries, VRML creation software needs to be developed that fully supports creation and use of PROTOs and EXTERNPROTOs. One of the goals of this research is to describe a hypothetical prototype for such a tool.

A tool designed to facilitate working with reusable components should reflect such designs in its interface, to encourage the user to also think in terms of modularity.

houdini

Houdini, from Side Effects Software Inc., is a high end procedural modeling and animation tool. What sets it apart from other modeling / animation packages is its use of data flow networks to represent construction processes (figure 4). Complex models and animations are built up procedurally by piping the output of of one primitive component into the input of another.

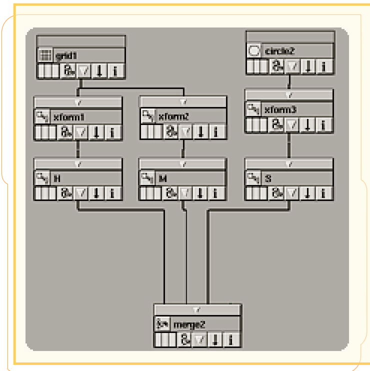


fig04: data flow network

Each component is a specialized, self-contained unit; a shape primitive, a transform operator, a particular modeling operation (e.g. extrusion), etc., essentially an emodel providing access to a simplified set of parameters for controlling a more complex process (figure 5). The components are iconified as little boxes, with name labels, desginated input and output areas, and flags to indicate termination of the network, in order to view the results accumulated up to that particular component.

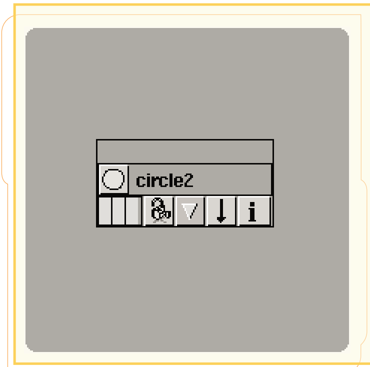


fig05: component

Components originate from a palette at the top of the screen, or from menus accessible via other components. They can be positioned anywhere in the workplace by dragging with the mouse, to allow the user to group them into meaningful arrangements. Clicking on the output of one component and the input of another compatible type forms a link between them, giving the latter access to the data output from the first component. Clicking in the main body of the component reveals its parameter interface, allowing modification of its characteristic values (figure 6). This is a highly flexible, very powerful form of visual programming, with a nominal learning curve. Its object oriented design allows for new functionality to be introduced easily in the form of new components, without requiring any changes to the components already existing or the way the user interacts with the interface.

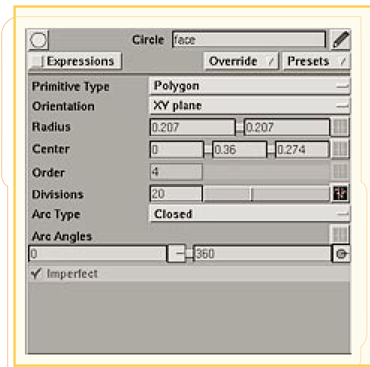


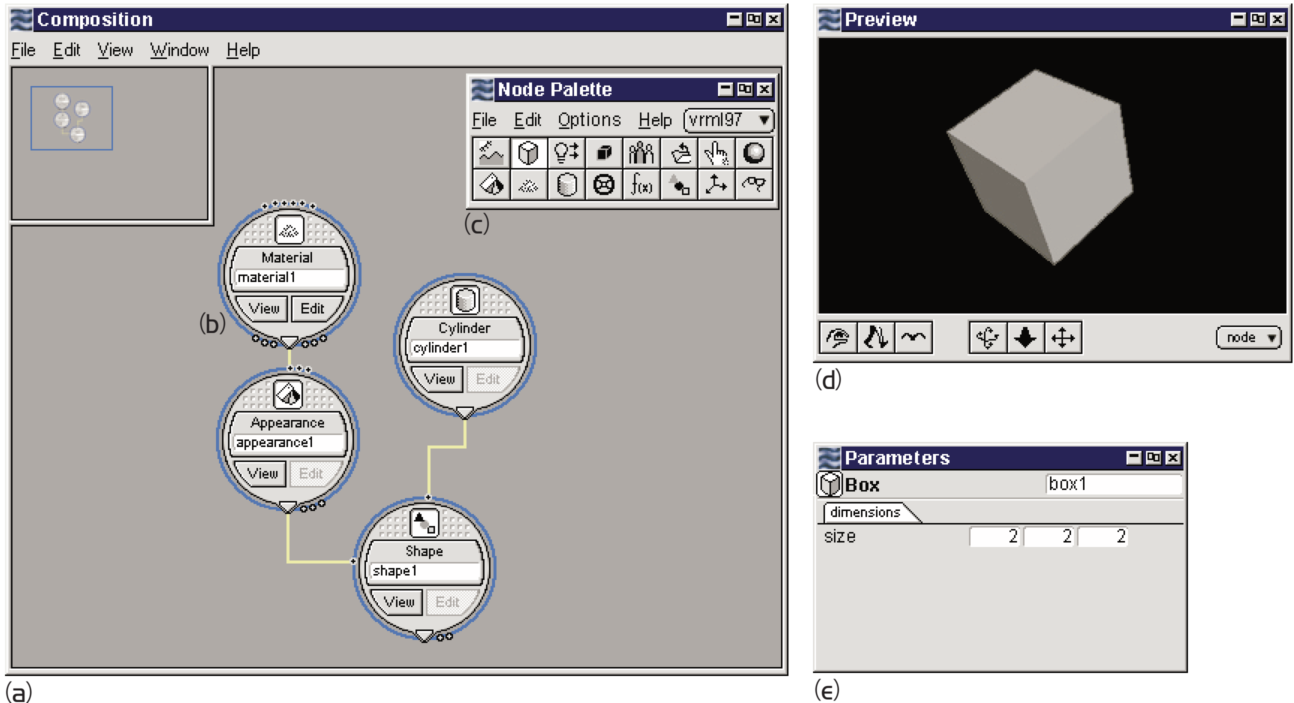
fig06: parameter window

Such a visual data flow network interface is highly applicable to the VRML creation process. It has been shown earlier in this paper that specialized VRML behaviors and utilities can be componentized into emodels with standardized input / output fields. A tool to allow visual networking and parameter adjusting of VRML emodels could provide the same power and flexibility to the VRML developer that Houdini provides to the 3D modeler / animator. Furthermore, novice VRML developers are often unaware or intimidated by the EXTERNPROTO construct in VRML. Providing these users with a tool that lets them visualize EXTERNPROTOS as simple building blocks empowers them to create products of greater complexity and quality without requiring them to become expert VRML users. However, expert users will benefit from a streamlined visual process designed especially to take advantage of reusable VRML components.

The tool described in this research is heavily influenced by the interface and workflow qualities of Side Effect's Houdini software. It is presented to illustrate the direct applicability of a similar visual data flow network interface to the VRML construction process. Houdini does support some subset of VRML through an export plugin, but the program as a whole is much more complex, expensive, and focused on different problems than a strictly VRML tool would be.

flux

The concept tool described here is codenamed **flux**, after the visual data flow paradigm it's modeled on. The core of **flux** is comprised of the following major areas:

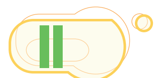


(a) **composition area:**	where components are arranged and linked
(b) **component tokens:**	the generic visual representations of components, for positioning and linking to others in the composition area.
(c) **component palette:**	where new components can be selected for placement in the composition area
(d) **view window:**	where the current node or entire scene can be seen
(e) **parameter window:**	where a specific component's parameters can be adjusted

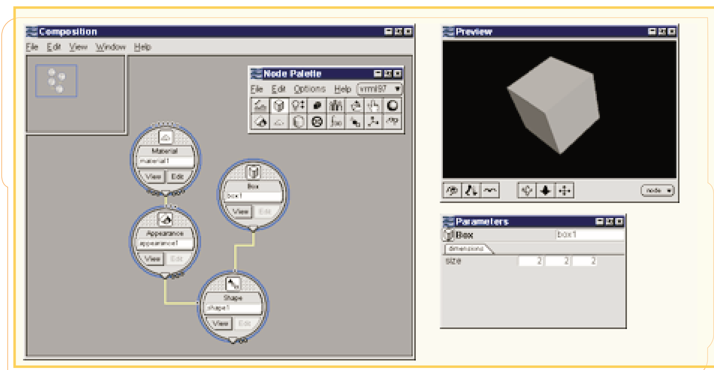
The strength of **flux** lies in the opportunity it provides the user to create a model and save it as a new component, which then becomes available from the component palette just as any of the standard VRML components. This process is simply a reflection of the EXTERNPROTO construct, which allows the creation and use of new node types referenced from an external file. However, **flux** simplifies and error-checks the tasks required to create, save and reference EXTERNPROTOS, at the same time providing a visual representation of the steps involved.

To illustrate how a program like **flux** would work, a walk-through is presented. It shows how new components would be created, using a simple button as example. A comparison of the visual process is made to the [non-visual] text file that would have to be written by hand to achieve the same results.

The example child's nightstand and clock presented earlier could be assembled using **flux**. A *RocketClock* component would be created first. This custom component would then be combined in a new file with other standard components to bring geometry, textures and lights into the scene. It is worthy of note that this sort of cumulative building can easily be continued for several generations, for instance saving the entire nightstand scene as another new component and using it inside a larger model of a house.

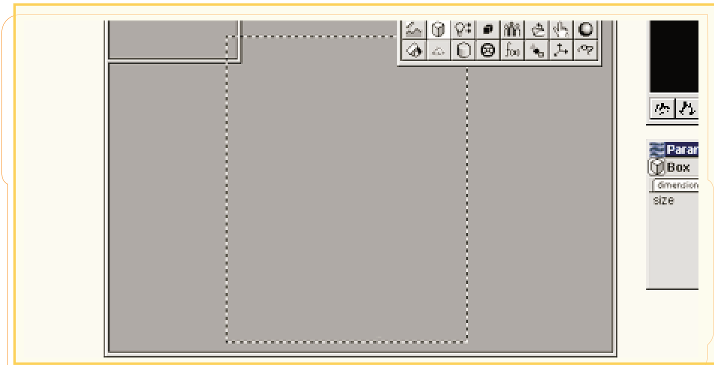


walk-through: creating new components



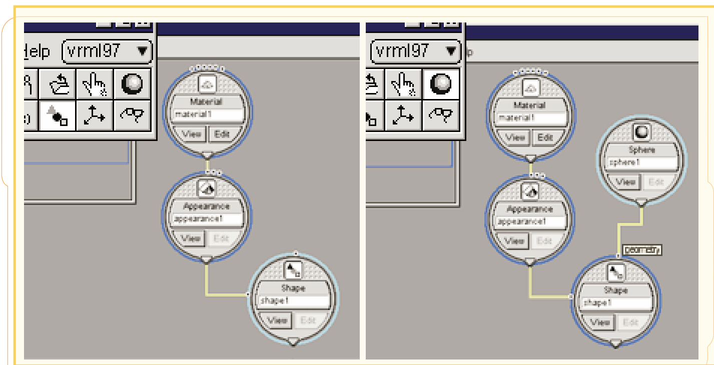
step01: startup

1] The **flux** program is started; it launches with the four core windows, and a simple default scene of a box.



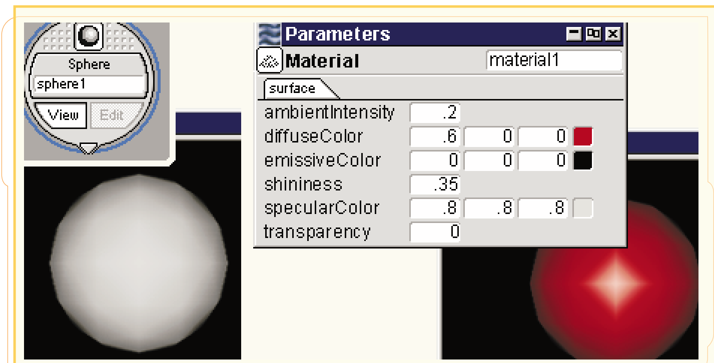
step02: clear the composition area

2] The default scene is selected and deleted by dragging a selection marquee around all the components and pressing the delete key to clear the component window.



step03: add geometry

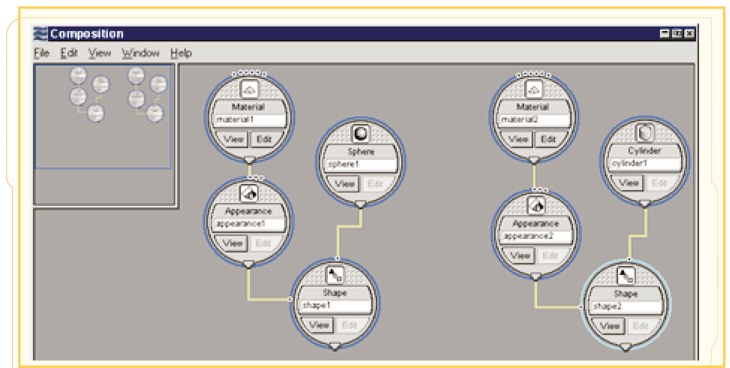
3] A Shape node is selected from the node palette and placed in the composition area. This automatically creates the necessary Appearance and Material nodes. To specify the geometry for the Shape, a Sphere is selected from the node palette, placed in the composition area, and linked to the *geometry* field of the Shape node. Field names and types are determined from pop-up notes that appear when the mouse is over one of the field/event dots around the node. Two fields or events can be linked together only if their data types match.



step04: adjust material properties

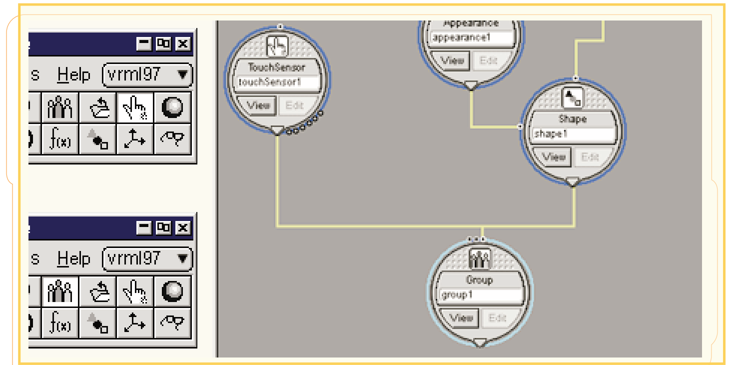
4] Clicking on the Sphere's view button shows a sphere in the view window, made of the default dull off-white material. To make the button shiny and red, the Material node is selected, and in the parameter window its *diffuseColor* and *shininess* values are adjusted.

5] Steps 3 and 4 are repeated, specifying a cylinder instead of a sphere. This will be the base of the button, so it is sized shorter along the y axis, larger in diameter and colored dull black.



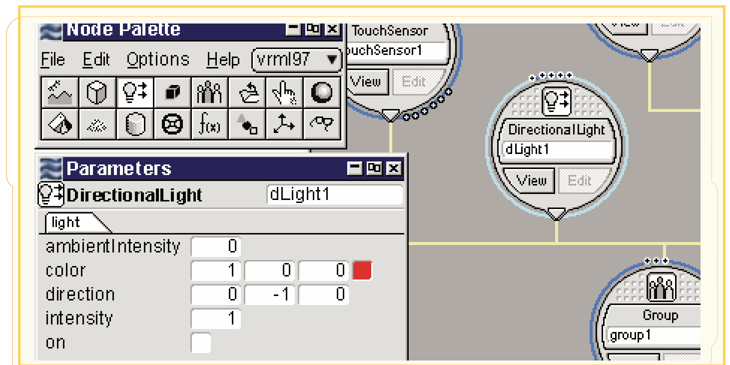
step05: create button base

6] Now a TouchSensor node is selected from the palette, as well as a Group node. The TouchSensor is a VRML node that monitors mouse picking events, such as mouseOver and mouseDown. It will be used to indicate when someone has clicked on the button. The Group node is required to limit the scope of the TouchSensor's awareness to just the button and not the base or any other geometry. The Group acts as a container; the TouchSensor will only be active for items within the Group. To place the TouchSensor and button Shape within the Group, their nodes are linked to the Group's *children* field.



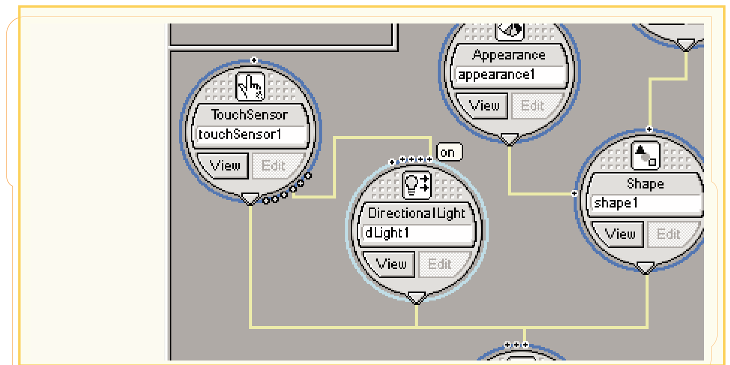
step06: add TouchSensor and container Group

7] There should be some form of feedback for the user clicking on the button. A light placed over the button could light it up at click time. A DirectionalLight is selected from the node palette, and placed in the composition area. From its parameter window, it is adjusted to point straight down the y axis, be bright red in color, and initially turned off. Since the light needs to shine on the button, and not on the base, it needs to be within the same Group, which will limit the range of its illumination. Simply linking the light to the *children* field of the Group already containing the TouchSensor and Shape nodes fulfills these requirements.

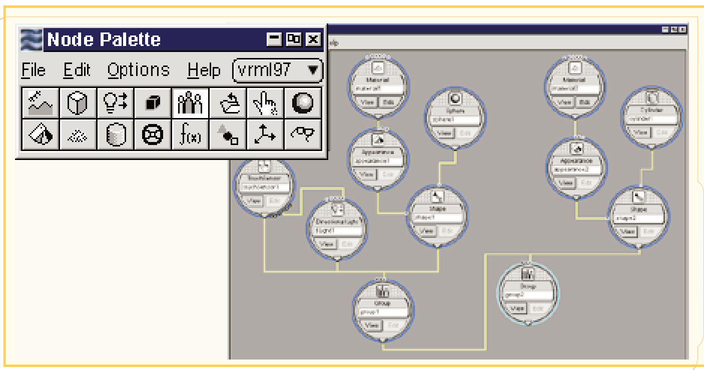


step07: add an indicator light

8] In order for the light to turn on, it needs to recieve a stimulus event. This will be provided by the *isActive* eventOut of the TouchSensor. Linking this to the *on* field of the DirectionalLight ensures the light will be activated when the button is clicked.

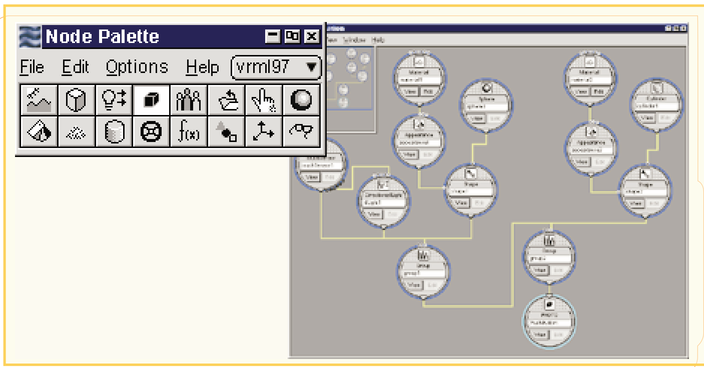


step08: link TouchSensor to DirectionalLight



step09: assemble final group

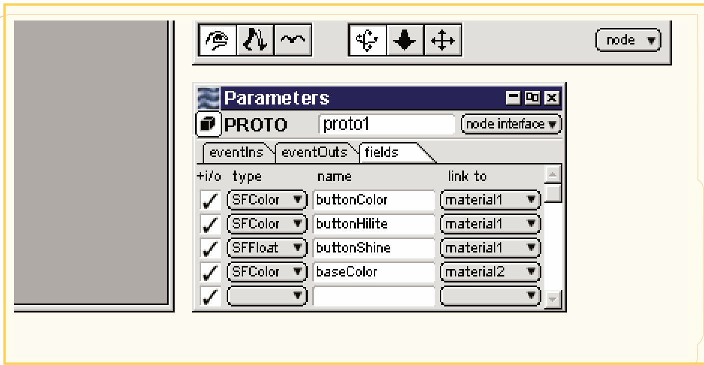
9] Since this button will become a component, all of its elements need to be grouped together (a VRML requirement). Another Group is added from the node palette, and the previous Group and button base are linked to its *children* field.



step10a: add PROTO

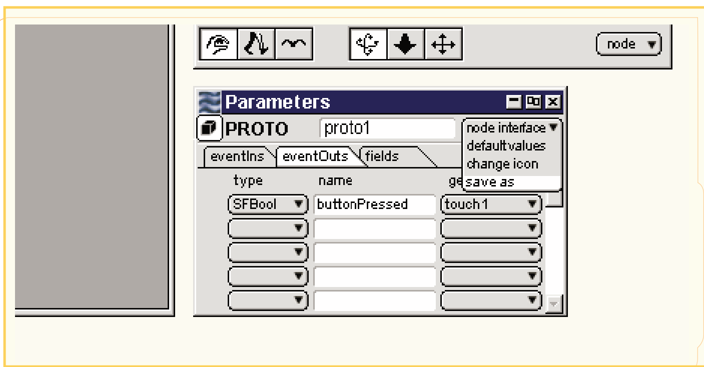
10] Finally, a PROTO is selected from the node palette and placed in the composition area. The Group containing all the button elements is linked to it. Now all that remains is to specify the properties of the button that will be configurable by future users of the component.

In the PROTO parameter window, the top pulldown menu is set to 'node interface'. This shows all the *eventIns* (input), *eventOuts* (output), and *fields* (stored data) for the component. Initially, none are defined.



step10b: define node interface

The component's base, button, and light colors and button shininess can be added to the PROTO interface by going to the 'fields' tab, providing names for them and choosing their value types from the pull down lists. To connect them to the actual base and button value fields in the model, the 'link to' drop-down is used, and the names of the appropriate base and button nodes are selected.



step10c: save component

The activation event resulting from a click on the button needs to be broadcast from the component. A new *eventOut* is added, and 'gotten from' the TouchSensor, touch1.

Lastly, the whole component needs to be saved to make it accessible from the node palette. This is slightly different than simply saving a file, since not all files are legitimate external components. Selecting 'save as' from the top drop-down menu brings up the component save dialog. After completing this step, the component is now ready to be selected from the

The finished button can be seen in figure 07, both off (unclicked), and on (clicked). Listing 01 shows the VRML source code for the example button, which without **flux**, would have to be generated by hand. This is not a daunting task for such a simple example, but as scene complexity increases, text files can quickly become large and hard to navigate within and between. Being able to encapsulate and componentize logical sections of the scene into visual tokens allows for much easier organization and management.

Having access to immediate visual feedback is also extremely valuable in a development environment, speeding up development time by eliminating the need to endlessly flip back and forth between text editor and browser window to see the results of a delicate value adjustment.

Additionally, a visual interface would eliminate most of the repetitive typing involved in creating a VRML file, by allowing drag-and-drop creation of nodes and intuitive parameter adjustment.

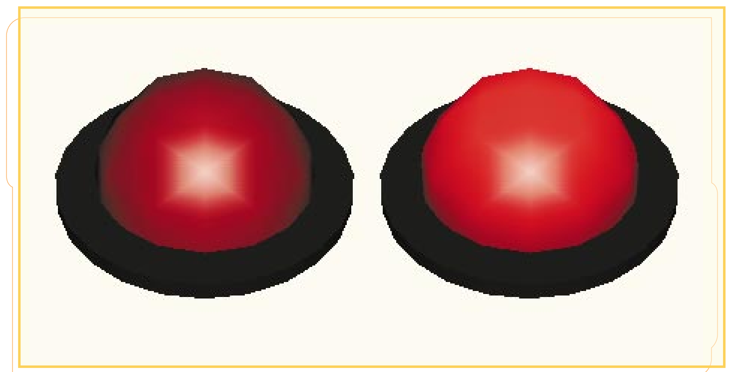


fig07: example button

```
#VRML V2.0 utf8

PROTO Button [
   exposedField SFColor buttonColor .6 0 0
   exposedField SFColor buttonHilite 1 0 0
   exposedField SFFloat buttonShine .35
   exposedField SFColor baseColor   .1 .1 .1
   eventOut     SFBool  buttonPressed
]
{

   Group {
      children [

         Group {
            children [

               DEF dLight1 DirectionalLight {
                  color  IS buttonHilite
                  direction 0 -1 0
                  on FALSE
               }

               DEF touch1 TouchSensor { isActive IS buttonPressed }

               Shape {
                  appearance Appearance {
                     material Material {
                        specularColor .8 .8 .8
                        diffuseColor IS buttonColor
                        shininess IS buttonShine
                     }
                  }
                  geometry Sphere {
                     radius .75
                  }
               }
            ]
         }

         Shape {
            appearance Appearance {
               material Material {
                  diffuseColor IS baseColor
                  shininess .8
               }
            }
            geometry Cylinder {
               height .125
            }
         }

      ]
   }
   ROUTE touch1.isActive TO dLight1.on

}

Button {}
```

listing01: button source code

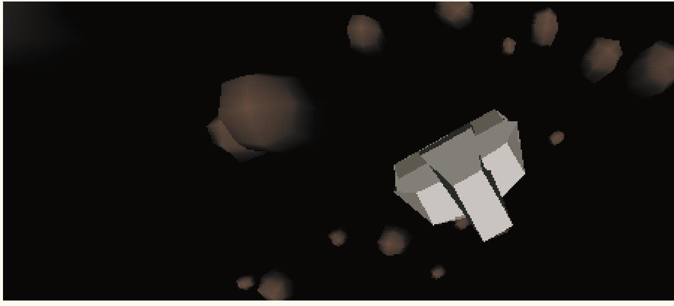


fig08: asteroids

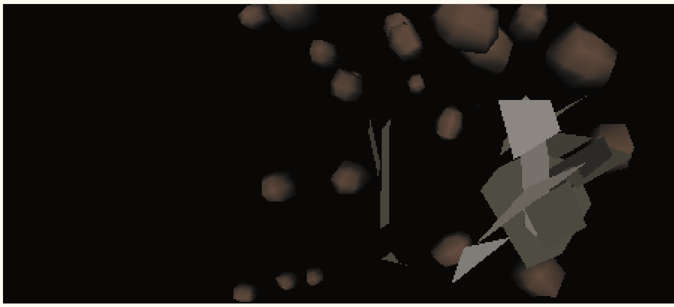


fig09: asteroids

fig10: asteroids

example game

As a proof of concept, an example game has been developed, using components from the library presented by this research. **Vrmloids** is patterned after Asteroids [Atari, 1979], a classic arcade game in which the player maneuvers a spaceship through an asteroid field, trying not to crash, and accumulating points for destroying the big rocks. Gameplay is simple but fun, and requires the implementation of several typical game behaviors, such as collision detection, explosions, tracking camera, keyboard input sensing, and a view-constant display (HUD).

Vrmloids is designed around three primitives: the ship, the rocks, and the ship's missiles. Collision detection must be performed between the ship and the rocks, as well as between the rocks and the missiles. Both the ship and rocks need to explode when a collsion is detected. The score needs to be displayed on the screen in the same location throughout the game. From the component library, *BSphereCollisionSensor*, *ExplodingPolyset*, *LookAt*, *KeyBoardSensor*, and *GlueToView* are used to address the main general behaviors of the game. Additional **Vrmloids**-specific code is required to connect the components together and perform other smaller tasks like updating the score and controlling the missiles.

Screenshots of the game can be seen in figures 08 through 10, and the source code is presented in Appendix C.

A listing of the resources presented by this research follows:

VRML components for games:

Component	Description
AvoidObstacles:	Autonomous obstacle avoidance
AxisJack:	Spatial reference
BSphereCollisionSensor:	Collision detection
ClockMechanism:	Rotating clock hands
ColorPicker:	Color choosing
Copy:	Object duplication
ElevationGridMaker:	Terrain creation
ExplodingPolySet:	Object explosion [65]
GetBBox:	Object bounding box computation
GetBSphere:	Object bounding sphere computation
GlueToView:	Positioning geometry relative to the viewpoint
Guide:	Autonomous pursuit / avoidance
Info:	Accessing object position / orientation properties
KeyboardSensor:	Keyboard input sensing [32]
Lookat:	Autonomous aiming
Ngon:	Polygon generation
Noise:	Noise generator
Output:	Dynamic content generation
RolloverSensor:	Mouse action capture
Stringerator:	(non-VRML) Text quoting, for dynamic content
VisibleLightsPackage:	Visible light parameters
VisibleSound:	Visible sound parameters
VisibleTransformsPackage:	Object manipulation
WaveInterpolator:	Sine / cosine interpolation

Detailed descriptions of the components can be found in Appendix B.

VRML component development environment prototype:

flux illustrates how a data-flow network interface might be applied to the VRML development process.

Example VRML game:

Vrmloids

Source code for the components and example game can be found in Appendix C.

summary

[12]

This research identifies VRML as a valid language for representing encapsulated models (as defined by May) in real-time 3D. Validation is presented through example implementation.

[15], [16]

Examination of video game design and development indicates the process could benefit significantly from the advantages of emodels, namely:

parameterization
reusability
increased complexity
decreased development time
componentization

[07], [08]
[21], [22], 23], [24], [79]

For the creating and playing of 3D games on the web, VRML is the strongest candidate, due to its open format, established user base, and broswer availability. There are currently a small but growing number of VRML creation tools available, although none are geared specifically towards game production. Similarly, there are some collections of reusable VRML components on the web, but none are specifically applicable to games.

[02], [03], [04], [05]

Despite the huge popularity of 3D games, and the large number of offline 3D games on the market, there is a very small number of VRML games. VRML game developers need better tools, and more expansive libraries of game related resources.

[29], [30], [31]

In the interest of promoting VRML game development, a library of emodels directly applicable to game production have been developed for use in VRML. Design of the emodels was based on an analysis of popular games and their common functionality, as well as an examination of the VRML development process, to provide tool for areas with the weakest support.

Amassing high-quality game oriented emodels will not be entirely helpful for the VRML game development community, however, without creation software support for emodels, namely the VRML EXTERNPROTO structure. To stimulate development in this area, an example VRML creation utility was prototyped to illustrate how such software might look and function.

To illustrate possible uses for the various emodels developed for this research, simple examples are provided for each component, and a more complex example game was created, using several different emodels.

future work

Through the course of this research, several areas were identified as interesting opportunities for future work. The development of a visual tool like **flux** for the procedural development of VRML components would be a boon to the Web3D community. A free texture application utility would complement the growing number of free 3D modelers, which typically don't offer robust texturing support. Such a utility could likely be built entirely in VRML. Finally, a library is never complete; new components and improvements over existing ones will remain an open opportunity for VRML developers as long as VRML has an audience.

NextGeneration

Imagine Publishing Inc. *NextGeneration*. September 1996.

Donkey Kong, 1981
Nintendo
ARCADE: Challenging Maneuvers of a Plumber

Command & Conquer, 1985
Virgin
REAL-TIME STRATEGY: Futuristic Warfare

World Series Baseball, 1995
Sega
SPORTS: Baseball

Worldwide Soccer 2, 1996
Sega
SPORTS: Soccer

Formula One Grand Prix 2, 1996
MicroProse
DRIVING: Formula One Racing

Outrun, 1986
Sega
DRIVING: Sports Car Racing

Spectre VR, 1993
Velocity
ACTION: Futuristic Tank Combat

Rolling Thunder, 1987
Namco
ACTION: Side scrolling fiction shooter

EF2000, 1995
Ocean
SIM: EF2000 Flight Combat

'Snake Game'
unknown
ARCADE: Challenging Maneuvers of a Snake

NHL Powerplay, 1996
Virgin
SPORTS: Hockey

Asteroids, 1979
Atari
ARCADE: Space Shooter

Zork I - III, 1980 - 1989
Infocom
TEXT ADVENTURE: Fantasy

Super Mario Kart, 1992
Nintendo
DRIVING: GoKart Racing

Rescue Raiders
360 Degrees / Sir-Tech
ACTION: 20th Century Fiction Warfare and Strategy

Duke Nukem 3D, 1996
3D Realms
ACTION: City Warfare

Nobanaga's Ambition, 1988
Koei
WARGAME: Medieval Japanese History

SimCity 2000, 1989 - 1995
Maxis
RESOURCE MANAGEMENT: City Building

Wipeout XL, 1996
Psygnosis
DRIVING: Futuristic Racing

Herzog Zwei, 1991
Sega
REAL-TIME STRATEGY: Fictional Warfare

Madden Football Series, 1991 - 1996
Electronic Arts
SPORTS: Football

Syndicate, 1993
Bullfrog
REAL-TIME STRATEGY: Futuristic Gangsterism

NFL Gameday, 1995
Sony CE
SPORTS: Football

Sam & Max Hit the Road, 1994
LucasArts
GRAPHIC ADVENTURE: Comic Book Character based

X-COM: UFO Defense, 1994
MicroProse
RPG: Alien vs. Human Warfare

Nights, 1996
Sega
ARCADE: Fantasy Flying / Racing

Lurking Horror, 1987
Infocom
TEXT ADVENTURE: College Campus Fiction

X-Wing / Tie Fighter, 1994
LucasArts
SIM: Star Wars Space Flight Combat

Tekken II, 1996
Namco
FIGHTING: Martial Arts

Daytona USA, 1994
Sega
DRIVING: Stock Car Racing

Sonic Series, 1991 - 1995
Sega
ARCADE: Challenging Maneuvers of a Hedgehog

Doom Series, 1993
Id Software
ACTION: Dungeon Combat

Micro Machines, 1991
CodeMasters
DRIVING: MicroMachine Racing

Final Fantasy Series, 1987 - 1995
SquareSoft
RPG: Fantasy

Populous Series, 1987 - 1992
Electronic Arts
RESOURCE MANAGEMENT: Civilization Building

Marble Madness:, 1984
Atari
ARCADE: Challenging Maneuvers of a Marble

Elite, 1982
FireBird
RESOURCE MANAGEMENT: Space Trade and Combat

Defender, 1981
Williams
ACTION: Space Shooter

Ms. Pac-Man, 1981
Midway / Namco
ARCADE: Challenging Maneuvers of a Female Circle

Virtua Racing, 1992
Sega
DRIVING: Formula One Racing

WarCraft II, 1995
Blizzard
REAL-TIME STRATEGY: Fantasy

Quake, 1996
Id Software
ACTION: Dungeon Combat

Lemmings, 1991
Psygnosis
ARCADE: Puzzle

Street Fighter 2, 1991
Capcom
FIGHTING: Martial Arts

Virtua Fighter 2, 1994
Sega
FIGHTING: Martial Arts

Mario Series, 1985 - 1991
Nintendo
ACTION: Fantasy Exploration

Civilization Series, 1994 - 1996
MicroProse
RESOURCE MANAGEMENT: Civilization Building

Super Bomberman 2, 1994
HudsonSoft
ACTION: Bomb Detonating

Tetris, 1987
Spectrum Holobyte
ARCADE: Block Stacking

Super Mario 64, 1996
Nintendo
ACTION: Fantasy Exploration

PC Gamer

Imagine Publishing Inc. *PCGamer*. May 1997.

The Elder Scrolls, Daggerfall
Bethesda Softworks
RPG: Fantasy

Pro Pinball: The Web
Empire / Interplay
SIM: Pinball

Tony LaRussa Baseball 3
Storm Front Studios
SPORTS: Baseball

Star Trail: Reals of Arkania
Sir-tech
RPG: Fantasy

Close Combat
Microsoft
WARGAME: 20th Century Warfare

V for Victory Series
Three-Sixty Pacific
WARGAME: WWII Warfare

Silent Hunter
SSI
SIM: WWII Submarine Combat

Might & Magic III: The Isles of Terra
New World Computing
RPG: Fantasy

Front Page Sports: Football Pro
Sierra
SPORTS: Football

Diablo
Blizzard Entertainment
RPG: Fantasy

Speedball 2: Brutal Deluxe
The Bitmap Brothers
SPORTS: Futuresport

Indiana Jones and the Fate of Atlantis
LucasArts
GRAPHIC ADVENTURE: Movie Character Based

SimCity 2000
Maxis
RESOURCE MANAGEMENT: City Building

D/Generation
Mindscape
ARCADE: Puzzle and action

Triple Play 97
Electronic Arts
SPORTS: Baseball

Monkey Island II: LeChuck's Revenge
LucasArts
GRAPHIC ADVENTURE: Pirates

Master of Orion
MicroProse
RESOURCE MANAGEMENT: Space Empire Building

Star Control II
Accolade
RESOURCE MANAGEMENT: Space Exploration and Combat

Wing Commander: The Kilrathi Saga
Origin
SIM: Futuristic Space Combat

Harpoon II
Three-sixty Pacific
SIM: 20th Century Naval Warfare

Lemmings
Psygnosis
ARCADE: Puzzle

Railroad Tycoon
MicroProse
RESOURCE MANAGEMENT: Railroad Building

The Complete Ultima VII
Origin
RPG: Fantasy

NHL '97
Electronic Arts
SPORTS: Hockey

Chuck Yeager's Air Combat
Electronic Arts
SIM: 20th Century Air Combat

Syndicate
Bullfrog
REAL-TIME STRATEGY: Futuristic Gangsterism

Beavis & Butthead in Virtual Stupidity
Viacom New Media
GRAPHIC ADVENTURE: Cartoon Character Based

Virtual Pool
Interplay
SPORTS: Pool

Alone In the Dark
I-Motion
ACTION: Haunted House

Populous
Electronic Arts
RESOURCE MANAGEMENT: Civilization Building

Ultima Underworld I & II
Origin Systems
ACTION: Dungeon Exploration

Gabriel Knight: Sins of the Fathers
Sierra
GRAPHIC ADVENTURE: 20th Century Fiction

Descent
Interplay
SIM: Futuristic Space Combat

EF2000
Ocean of America
SIM: EF2000 Air Combat

Duke Nukem 3D
3D Realms
ACTION: City Combat

Quake
ID Software
ACTION: Dungeon Combat

Tomb Raider
Eidos Interactive
ACTION: Tomb Raiding

Panzer General
SSI
WARGAME: WWII Warfare

Red Baron
Sierra/Dynamix
SIM: WWI Flight Combat

Sam & Max Hit the Road
LucasArts
GRAPHIC ADVENTURE: Comic Book Character based

AH-64D Longbow
Jane's Combat Simulations
SIM: AH-64D Longbow Helicopter Combat

Links LS
Access
SPORTS: Golf

Command & Conquer: Red Alert
Virgin/Westwood Studios
REAL-TIME STRATEGY: European Warfare

X-COM: UFO Defense
MicroProse
RPG: Alien vs. Human Warfare

Heroes of Might & Magic II
New World Computing
RPG: Fantasy

System Shock
Origin
ACTION: Space Fantasy Action

Civilization II
MicroProse
RESOURCE MANAGEMENT: Civilization Building

WarCraft II
Blizzard
REAL-TIME STRATEGY: Fantasy

Doom
ID Software
ACTION: Dungeon Combat

Tie-Fighter Collector's CD-ROM
LucasArts
SIM: Star Wars Space Flight Combat

PC Games

IDG Communications. *PC Games*. August 1998.

X- Wing
You Don't Know Jack
System Shock
Pirates!
Might and Magic
AH-64 Longbow 2
Gabriel Knight: The Beast Within
Sam & Max Hit the Road
X-Wing vs. TIE Fighter
Dungeon Keeper
Wasteland
Betrayal at Krondor
NASCAAR Racing 2
FIFA 98
Master of Orion III
Privateer
Daggerfall
Curse of Monkey Island
Bard's Tale
Interstate '76
Wolfenstein 3D
Tetris
MULE
SimCity 2000
NHL 98
Descent
Heroes of Might and Magic II
Riven
Star Control II
Secret of Monkey Island
Myth: The Fallen Lords
TIE Fighter
Ultima IV
Myst
Command & Conquer

X-COM: UFO Defense, 1994
Microprose
RPG: Alien vs. Human Warfare

Tomb Raider, 1996
Eidos Interactive
ACTION: Tomb Raiding

MechWarrior 2, 1995
Activision
SIM: Futuristic Robot Combat

Doom II, 1994
Id Software
ACTION: Dungeon Combat

Diablo, 1996
Blizzard Entertainment
RPG: Fantasy Action

Duke Nukem 3D, 1996
3D Realms
ACTION: City Combat

Command & Conquer: Red Alert, 1996
Westwood Studios
REAL-TIME STRATEGY: 20th Century European Warfare

Civilization, 1993
Microprose
RESOURCE MANAGEMENT: Civilization Building

Doom, 1993
Id Software
ACTION: Dungeon Warfare

Quake, 1996
Id Software
ACTION: Dungeon Warfare

Jedi Knight, 1997
LucasArts
ACTION: Star Wars Combat

Civilization II, 1996
Microprose
RESOURCE MANAGEMENT: Civilization Building

Warcraft II, 1995
Blizzard Entertainment
REAL-TIME STRATEGY: Fantasy

Quake II, 1997
Id Software
ACTION: Futuristic Ruins Combat

Total Annihilation, 1997
Cavedog Entertainment
REAL-TIME STRATEGY: Futuristic Warfare

AxisJack

A reference device, simply a static prop indicating where the VRML world origin (0, 0, 0 in coordinate space) is located. When composing a scene in 3D space, it is often useful to refer to orientation cues such as this.

This file can be inlined for developmental purposes, and removed for the final version.

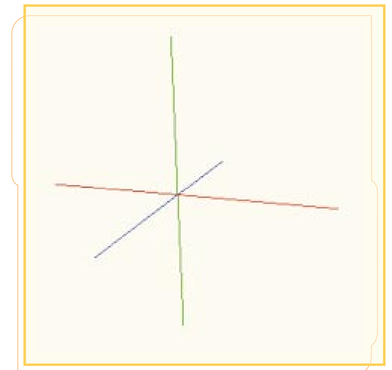


fig11: AxisJack

AvoidObstacles

Given a current position, a target position, a list of obstacle fields, and a velocity, generates a new position closer to the target position without intersecting the obstacle fields.

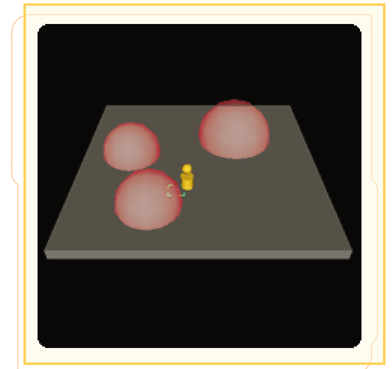


fig12: AvoidObstacles

BSphereCollisionSensor

Detects collisions between a subject and a group of obstacles. The collision time and obstacle id are reported. Both the subject and obstacles nodes are required to have position and bounding sphere radius fields. No collision detection is performed amongst the obstacles.

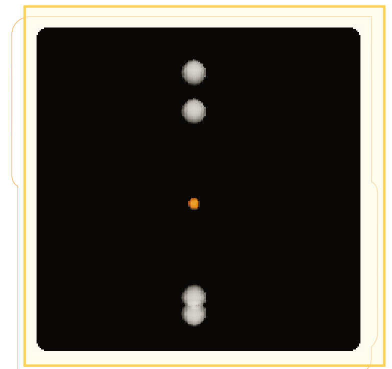


fig16: BSphereCollisionSensor

ColorPicker

Utility for choosing colors in VRML. Colors are selected using a Hue / Saturation / Brightness color model, and their corresponding Red, Green, and Blue values are printed on the screen.

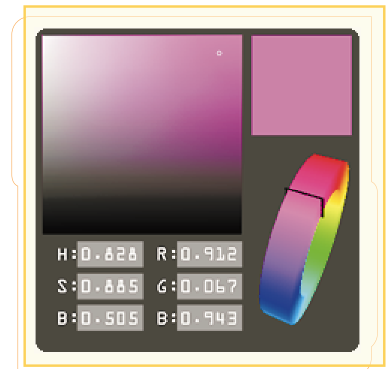


fig18: ColorPicker

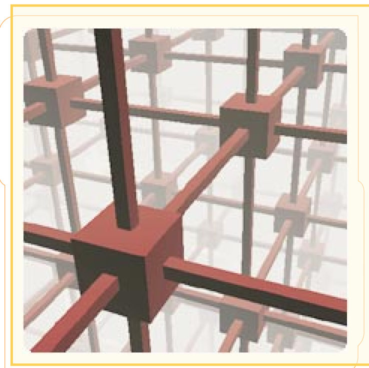


fig19: Copy

Copy

Makes duplicate instances of the child geometry, applying translation, rotation, or scale transformations cumulatively.

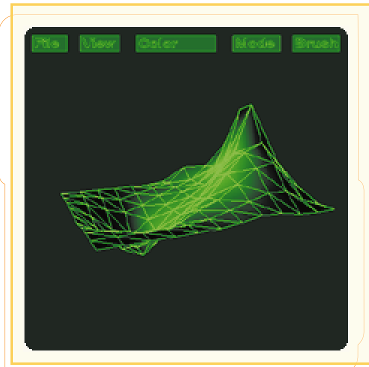


fig21: EvGridMaker

EvGridMaker

Utility to create elevation grids in VRML. The user ‘paints’ height information onto a arbitrarily sized grid.

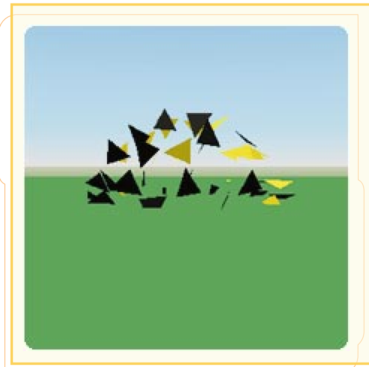


fig22: ExplodingPolySet

ExplodingPolySet

Original code from Bulatov [65]. Splits an indexed face set into individual polygons and then scatters the polygons from a central point when triggered, applying gravitational acceleration.

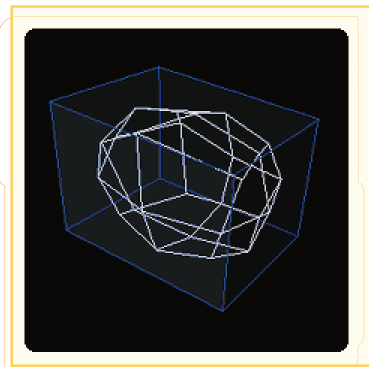


fig13: GetBBox

GetBBox

Computes the smallest rectangular volume that will enclose all of the object’s geometry. The bounding box center and size values are printed to the browser console, for use with nodes that use bounding box parameters (e.g. any of the VRML grouping nodes). The bounding box edges are parallel to the global coordinate axes.

GetBSphere

Computes a close approximation of the smallest spherical volume that will enclose all of the object's geometry (accurate to within +5%). The bounding sphere center and radius values are printed to the browser console, for use with nodes that use bounding sphere parameters (e.g. BoundingSphereCollisionSensor).

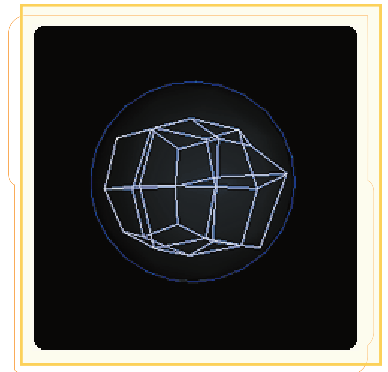


fig14: GetBSphere

GlueToView

Holds geometry in a fixed position relative to the user's viewpoint. Useful for Heads Up Displays.

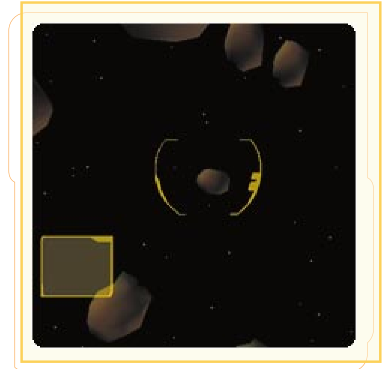


fig26: GlueToView

Guide

Maneuvers its children towards or away from a provided goal position. The goal may be any target moving through 3D space. The chase / flee state is set with a boolean switch.

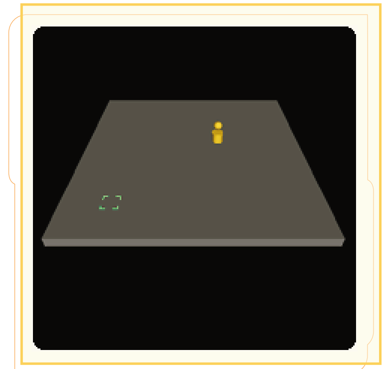


fig17: Guide

Info

An information tool for generating direction/orientation/position related nodes. The user navigates to the desired position, 'looks' in the desired direction, and presses a button to print out the appropriate node information. Nodes supported (clockwise from top right): Viewpoint, Transform, Sound, PointLight, DirectionalLight, SpotLight. The tool can be repositioned by dragging on the large 'i'.

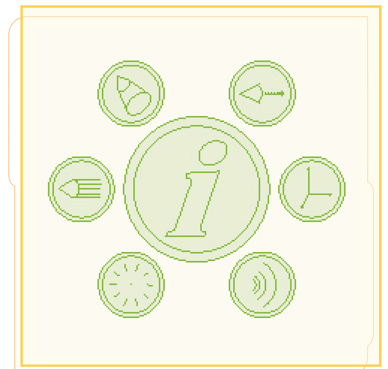


fig27: Info

fig28: InRangeSensor

KeyBoardSensor

Adapted from Joel [32], KeyBoardSensor uses a small java applet which communicates through a specified node interface to the VRML scene. The scene is embedded in an html page with the java applet, which acts as a go-between to capture keyboard events generated on the webpage and transmit them into the vrml scene. The keys are identified by their ASCII values.

fig29: LookAt

LookAt

Given a current position, look direction and a target position, generates a rotation to re-orient from the current look direction to look at the target.

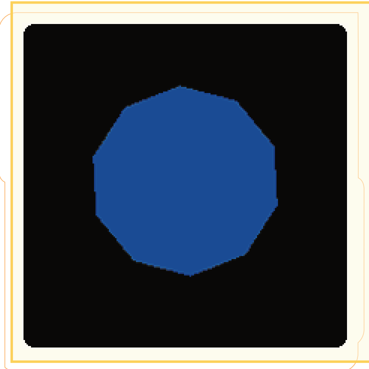


fig30: Ngon

Ngon

Generates an n-sided polygon or line set.

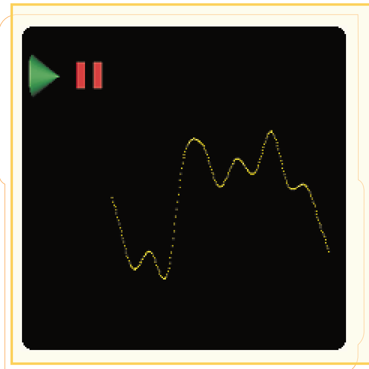


fig31: Noise

Noise

Noise generator, as described and implemented by Peachey [75]. See p.07 for a description of noise and its uses.

Output

Prints a stream of text with a specified MIME type to a new window. Provides parameters to control the new window characteristics, as in Javascript / HTML.

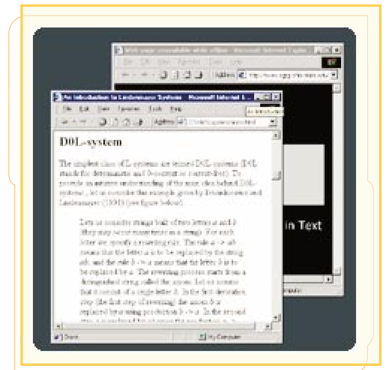


fig32: Output

RolloverSensor

Switches between geometries based on mouse input. Supports three states: default, mouseOver, and mouseDown, and can be enabled or disabled. Output consists of boolean values indicating mouseOver and leftClick actions, as well as leftClick time and the point on the object under the mouse cursor.

fig33: RolloverSensor

Stringerator

Javascript utility to convert plain text to quoted text for use with dynamic creation methods, such as CreateVrmlFromString.

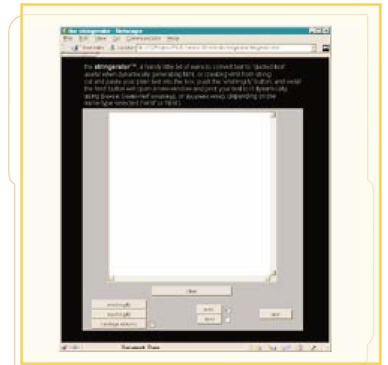


fig34: Stringerator

VisibleDirectionalLight

A graphical representation of the VRML DirectionalLight, indicating color, direction, and intensity.

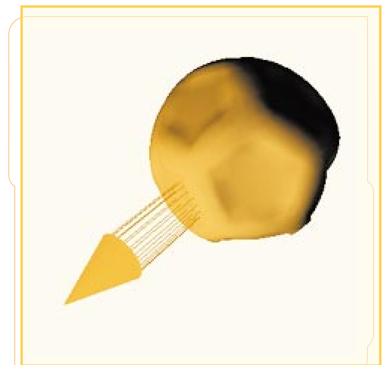


fig36: VisibleDirectionalLight

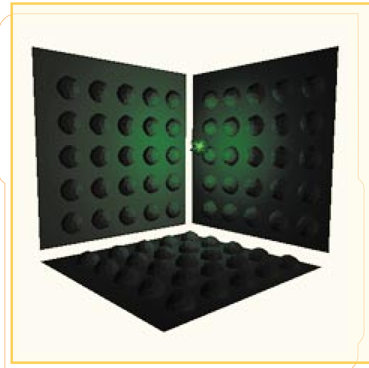


fig35: VisiblePointLight

VisiblePointLight

A graphical representation of the VRML PointLight, indicating color, intensity, location, and radius.

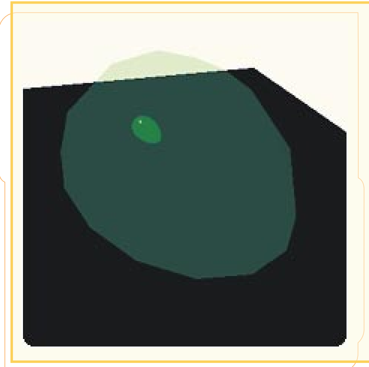


fig37: VisibleSound

VisibleSound

A graphical representation of the geometric volumes for the minimum and maximum intensities of a sound.

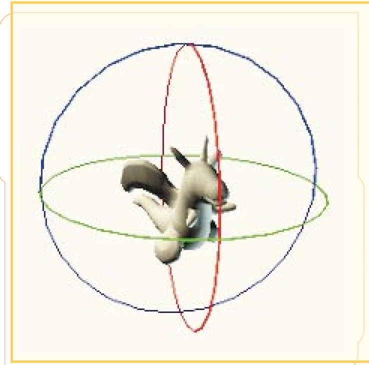


fig38: VisibleRotate

VisibleRotate

A visual rotation group node. Objects to be rotated are included in the VisibleRotate's children field. Degrees of rotation for each axis are printed to the VRML console and browser's status bar.

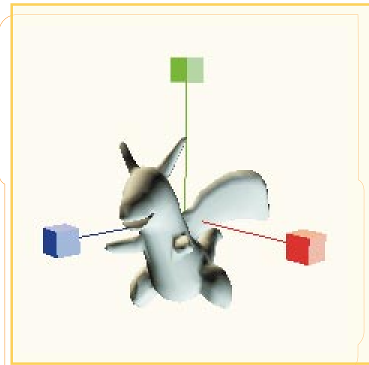


fig39: VisibleScale

VisibleScale

A visual scale group node. Objects to be scaled are included in the VisibleScale's children field. Scale values for each axis are printed to the VRML console and browser's status bar.

VisibleTranslate

A visual translation group node. Objects to be translated are included in the VisibleTranslate's children field. Translation values for each axis are printed to the VRML console and browser's status bar.

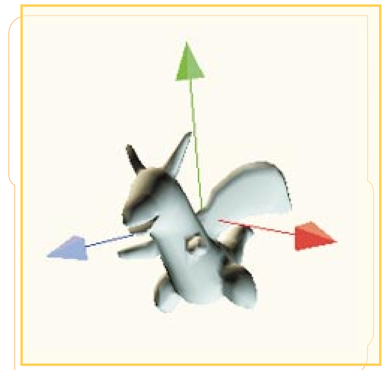


fig40: VisibleTranslate

WaveInterpolator

When given a fractional value of time, from 0 to 1, outputs the corresponding value of sine or cosine. Can be normalized to output values in the range of 0 to 1 instead of -1 to 1. Can also return the absolute value of sine or cosine.

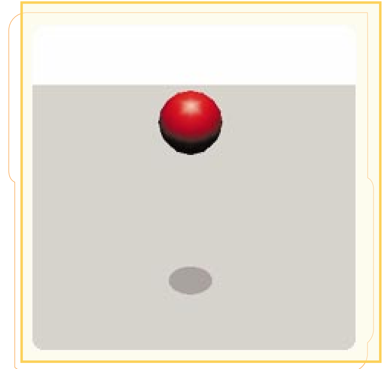


fig41: WaveInterpolator

RocketClock

```
rocketClock.wrl:
#VRML V2.0 utf8

EXTERNPROTO RocketClock [
   eventIn      SFVec3f  set_alarm
   exposedField SFColor  rocketColor
   exposedField SFColor  specularColor
   exposedField SFFloat  shininess
   exposedField MFString alarmSound
   exposedField SFFloat  alarmPitch
   exposedField SFFloat  alarmVolume
   exposedField SFFloat  shadowTransparency
   field        SFFloat  turbulence
   field        SFColor  lightColor
   field        SFBool   autoClock
   field        SFVec3f  time
   field        SFVec3f  alarm
   field        SFBool   alarm_enabled
] "RocketClockPROTO.wrl"

## Scene setup
Background { skyColor [ .06 .12 .24 ] }

NavigationInfo {
   headlight FALSE
   type [ "EXAMINE", "ANY" ]
}

Viewpoint {
   position -0.53 0 1.428
   orientation 0 -1 0 0.379
   description " blast off"
   jump FALSE
}

DirectionalLight { # -- high light
   color .2 .4 .8
   intensity .75
   direction 0.844 0.46 0.276
}

## Scene
Group { # --soft lit wall and tabletop
   children [
      DirectionalLight {
         color .2 .4 .8
         direction 0 -1 -1
         intensity .5
      }
      Transform { # wall
         scale 2.5 2.5 2.5
         rotation 0 1 0  3.9
         translation 1 0 -3
         center -1 0 0
         children [ Inline { url [ "wall.wrl" ] } ]
      }
      Transform {
         scale .5 .5 .5
         children [ Inline { url [ "tableTop.wrl" ] } ]
      }
   ]
}

DEF S Script {
   eventOut SFVec3f alarm_changed
   url ["javascript:
      function initialize() {
         var d = new Date();
         var hr = d.getHours();
         hr = (hr > 12) ? hr - 12 : hr;
         var m = d.getMinutes();
         var s = d.getSeconds() + 15;
         if(s >= 60) { s = 0; m++; }
         alarm_changed = new SFVec3f(hr, m, s);
      }
      "]
}

Group {
    children [
     DirectionalLight { # --fill light
         color .95 .98 1
         intensity .75
         direction -0.861 -0.333 0.384
      }
      DEF R RocketClock {
         alarmSound [ "blastoff_25.wav" ]
         alarmPitch .15
         shadowTransparency .7
     }
   ]
}

ROUTE S.alarm_changed TO R.set_alarm
```

```
RocketClockPROTO.wrl:
#VRML V2.0 utf8

PROTO RocketClock [
   eventIn      SFVec3f  set_alarm
   exposedField SFColor  rocketColor     .5 .5 .5
     exposedField SFColor  specularColor      0.6 0.6 0.6
     exposedField SFFloat  shininess            0.3
   exposedField MFString alarmSound      []
   exposedField SFFloat  alarmPitch      .7
   exposedField SFFloat  alarmVolume     1
   exposedField SFFloat  shadowTransparency 0
   field        SFFloat  turbulence      .6
   field        SFColor  lightColor      1 .6 .2
   field        SFBool   autoClock       TRUE
   field        SFVec3f  time            0 0 0
   field        SFVec3f  alarm           12 0 0
   field        SFBool   alarm_enabled   TRUE
]
{

   # Protos
   EXTERNPROTO ClockMechanism [
      eventIn  SFVec3f    set_time
      eventIn  SFVec3f    set_alarm
      eventIn  SFBool     set_alarm_enabled
        field    SFBool     autoClock
      field    SFVec3f    time
      field    SFVec3f    alarm
      field    SFBool     alarm_enabled
        field    SFVec3f    hourHandAxis
        field    SFVec3f    minuteHandAxis
        field    SFVec3f    secondHandAxis
        field    SFBool     twentyFourHours
      eventOut SFVec3f    time_changed
        eventOut SFRotation hourHand_changed
        eventOut SFRotation minuteHand_changed
        eventOut SFRotation secondHand_changed
      eventOut SFTime     alarmTime
   ] "ClockMechanismPROTO.wrl"

   EXTERNPROTO Noise [
      eventIn  SFVec3f set_vec3
      eventIn  SFFloat set_scale
      eventIn  SFFloat set_offset
      field    SFFloat scale
      field    SFFloat offset
      eventOut SFFloat value_changed
   ] "noisePROTO.wrl"


   Group {
      children [

         Transform {
            scale .5 .5 .5
            children [

               DEF RUMBLE Transform {
                  children [

                     Sound { # -- alarm sound
                        intensity IS alarmVolume
                        source DEF A AudioClip {
                           pitch IS alarmPitch
                           url IS alarmSound
                        }
                     }
                     Group { # -- rocket body w/ reflected light
                        children [
                           DEF LIGHTref DirectionalLight {
                              on FALSE
                              color .9 .4 0
                              intensity .4
                              direction 0 1 0
                           }
                           Inline { url [ "rocketBody.wrl" ] } ## this is to save space, in RocketClockPROTO
                                                               ## the geometry would be here and IS-mapped
                           Inline { url [ "rim.wrl" ] }        ## to save space

                        ]
                     }
                     Group { # -- clock face, w/ face light
                        children [
                           DEF LIGHTface DirectionalLight { color .4 .4 .4 }
                           Inline { url [ "face.wrl" ] }       ## to save space
                        ]
                     }
                     Group { # -- hands with front light
                        children [
                           DirectionalLight { intensity .65 }
                           DEF H Transform { # -- hour hand
                              center .001 .359 .28
                              children [ Inline { url [ "hour.wrl" ] } ] ## to save space
                           }
                           DEF M Transform { # -- minute hand
                              center .001 .359 .285
                              children [ Inline { url [ "minute.wrl" ] } ] ## to save space
                           }
                           DEF S Transform { # -- second hand
                              center .001 .359 .291
                              children [ Inline { url [ "second.wrl" ] } ] ## to save space
                           }
                        ]
                     }
```

```
                     Transform { # -- engine glow
                        translation 0 -.3 0
                        children [
                           DEF LIGHTeng SpotLight {
                              on FALSE
                              direction 0 -1 0
                              beamWidth .3
                              radius 2
                              color .9 .4 0
                           }
                        ]
                     }
                  ]
               } # -- end rumble
            ]
         }
      ]
   }
   # mechanics
   DEF C ClockMechanism { # -- spin the clock hands
      set_alarm IS set_alarm
      alarm_enabled IS alarm_enabled
      autoClock IS autoClock
      time IS time
      alarm IS alarm
   }
   DEF T TimeSensor { # -- drive the seed generator
      cycleInterval 10
      loop TRUE
      startTime -1
   }
   DEF P PositionInterpolator { # -- seed values for noise function; 10 gives good spread
      key      [ 0 .5 1 ]
      keyValue [ 0 0 0, 10 10 10, 0 0 0 ]
   }
   DEF P1 PositionInterpolator { # -- positions to index into using noise value
      key      [ 0 .11 .22 .33 .44 .55 .66 .77 .88 1 ]
      keyValue [ 0 0 0, 1 0 0, 1 0 1, -1 0 0, -1 0 -1, -1 0 1, 0 0 -1, 1 0 -1, 0 0 1, 0 0 0 ]
   }
   DEF N Noise { scale .5 offset .5 } # -- rrrumble
   DEF D Script {
      eventIn  SFTime  activateLights
      eventIn  SFFloat set_intensity
      eventIn  SFVec3f set_vec3
      eventIn  SFBool  soundPlaying
      field    SFNode  liteF USE LIGHTface
      field    SFNode  liteE USE LIGHTeng
      field    SFNode  liteR USE LIGHTref
      field    SFColor liteF_on  IS lightColor
      field    SFColor liteF_off .4 .4 .4
      field    SFFloat exxag IS turbulence
      field    SFNode  audio USE A
      field    SFNode  timer USE T
      field    SFBool  soundOn FALSE
      eventOut SFVec3f vec3_changed
      eventOut SFVec3f alarm_changed
      eventOut SFBool  alarmOn
      directOutput TRUE
      mustEvaluate TRUE
      url ["javascript:
         function shutOff() {
            liteR.set_on = FALSE;
            liteF.set_color = liteF_off;
            liteE.set_on = FALSE;
            alarmOn = FALSE;
            soundOn = FALSE;
            timer.set_loop = FALSE;
            timer.set_enabled = FALSE;
         }
         function activateLights(time) {
            liteR.set_on = TRUE;
            liteF.set_color = liteF_on;
            liteE.set_on = TRUE;
         }
         function set_intensity(val) {
            var dim = ((val + 1) / 4) + .1; // .1 - .6
            var bri = dim + .4; // .5 - 1
            liteR.intensity = dim;
            liteF.intensity = bri;
            liteE.intensity = bri;
         }
         function set_vec3(val, time) { vec3_changed = val.multiply(exxag / 10); }
         function soundPlaying(bool) {
            if(bool && !soundOn) { soundOn = TRUE; }
            if(!bool && soundOn) { shutOff(); }
         }
         "]
   }
   # move hands of clock
   ROUTE C.hourHand_changed   TO H.set_rotation
   ROUTE C.minuteHand_changed TO M.set_rotation
   ROUTE C.secondHand_changed TO S.set_rotation
   # at alarm time, start timer, sound, and turn on lights
   ROUTE C.alarmTime          TO A.set_startTime
   ROUTE C.alarmTime          TO T.set_startTime
   ROUTE C.alarmTime          TO D.activateLights
   # track sound progress with isActive -- timestamps unreliable
   ROUTE A.isActive           TO D.soundPlaying
   # convert timer fraction to SFVec3f, pass to noise,
   # scale noise by turbulence and wiggle rocket
   ROUTE T.fraction_changed   TO P.set_fraction
   ROUTE P.value_changed      TO N.set_vec3
   ROUTE N.value_changed      TO P1.set_fraction
   ROUTE N.value_changed      TO D.set_intensity
   ROUTE P1.value_changed     TO D.set_vec3
   ROUTE D.vec3_changed       TO RUMBLE.set_translation
}
```

Source CD-ROM

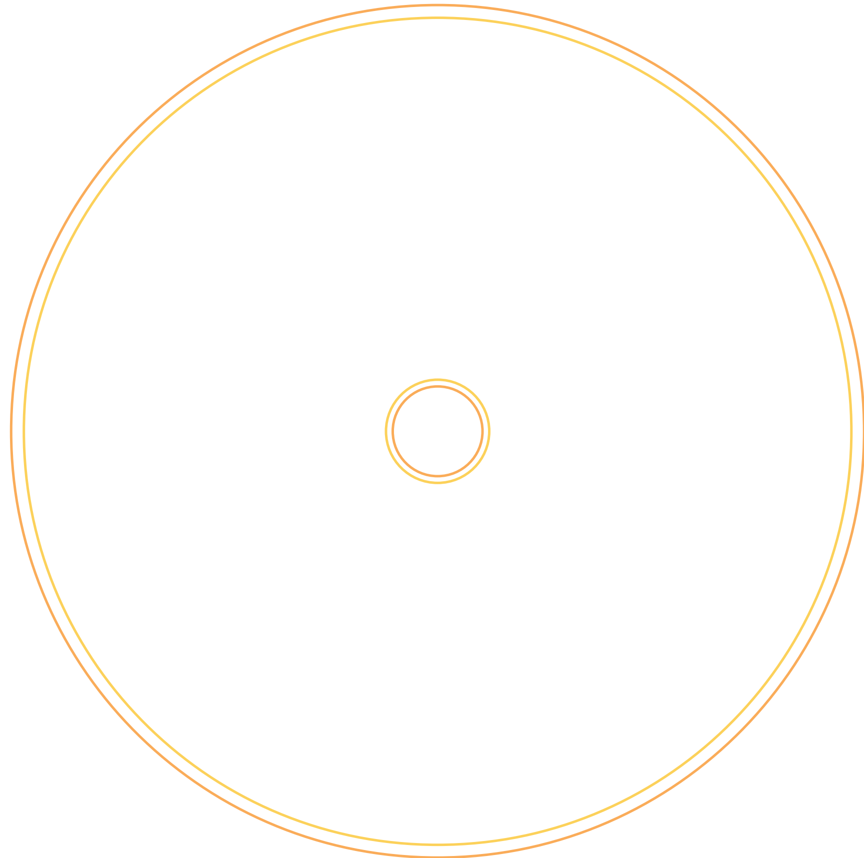
To save paper and space, the source code for the components and example games developed for this research is presented here on CD-ROM. To access the code listings, follow the instructions below:

For Windows:	insert cd and the web page should automatically launch. If not, view the contents of the cd, find index.html in the root directory, and open it in a web browser.
For Mac:	Double-click on the cd icon to view its contents, and double-click on index.html to open it in a web browser.
For Unix:	Open cdrom/index.html in a web browser.

Source Website

For those reading this paper electronically, the source code for the components is also available at:
http://www.cgrg.ohio-state.edu/~pgerstma/protolib/

Source for the example game is available at:
http://www.cgrg.ohio-state.edu/~pgerstma/vrml/



[01] The VRML Consortium Incorporated. *VRML97 International Standard*. http://www.web3d.org/fs_specifications.htm, 1997.

[02] Calvine Chew. *Pacman*. http://yallara.cs.rmit.edu.au/~chcc/Pacman/, 1998.

[03] Steve Guynup. *Media Assassin*. http://www.pd.org/~thatguy/games/mainindex.html, 1999.

[04] Joe Dellinger, Nikita Mikros, Flying Mikros Interactive. *ChickTacToe*. http://www.idfx.com/chicktactoe/, 1997.

[05] Sony Entertainment. *Mask of Zorro*. http://e3000-1.spe.sony.com/movies/zorro/javagameserver/level1.html, 1998.

[06] Satoshi Konno. *CyberToolbox*. http://www.cyber.koganei.tokyo.jp/top/index.html, 1999.

[07] Parallel Graphics, *Internet Space Builder*. http://www.parallelgraphics.com, 1999.

[08] Virtock Technologies. *Spazz3D*. http://www.spazz3D.com, 1999.

[09] Discreet. *3D Studio MAX R3*. http://www.ktx.com, 1999.

[10] Alias/Wavefront. *Maya*. http://www.aliaswavefront.com/entertainment/solutions/about_maya/index.html, 1999.

[11] Side Effects Software, Inc. *Houdini 4.0*. http://www.sidefx.com/, 1999.

[12] Stephen Forrest May. *Encapsulated Models: Procedural representations for Computer Animation*. The Ohio State University, March 1998.

[13] Matt Lewis. *Parameterized Face*. http://www.cgrg.ohio-state.edu/~mlewis/VRML/Class/w6/face.wrl, 1998.

[14] Stephen Forrest May. *Encapsulated Models: Procedural representations for Computer Animation*. p.24. The Ohio State University, March 1998.

[15] Stephen Forrest May. *Encapsulated Models: Procedural representations for Computer Animation*. p.23. The Ohio State University, March 1998.

[16] Stephen Forrest May. *Encapsulated Models: Procedural representations for Computer Animation*. p.121. The Ohio State University, March 1998.

[17] The VRML Consortium Incorporated. *VRML97 International Standard: Section 4.12, Scripting*. http://www.web3d.org/technicalinfo/specifications/vrml97/part1/concepts.html#4.12, 1997.

[18] The VRML Consortium Incorporated. *VRML97 International Standard: Section 4.9, External Prototype Semantics.* http://www.web3d.org/technicalinfo/specifications/vrml97/part1/concepts.html#4.9, 1997.

[19] Stephen Forrest May. *Encapsulated Models: Procedural representations for Computer Animation*. p.124. The Ohio State University, March 1998.

[20] Stephen Forrest May. *Encapsulated Models: Procedural representations for Computer Animation*. p.135. The Ohio State University, March 1998.

[21] VRML Proto Repository. http://www.vrml-content.org/, 1999.

[22] VRML Developer's Library. http://www.vapourtech.com/vrmlguide/index.html, 1999.

[23] Roland Smeenk. *Roland's VRML97 Site, VRML worlds and examples*. http://www.a1.nl/homepages/rsmeenk, 2000.

[24] Braden N. McDaniel. *Endoframe - VRML PROTOs*. http://www.endoframe.com/vrml/protos/index.html, 2000.

[25] Peter Duffett-Smith. *Practical Astronomy with Your Calculator, second ed.* Cambridge University Press, 1985.

[26] Albert E. Waugh. *Sundials, Their Theory and Construction*. Dover Publications, Inc., 1973.

[27] Frank W. Cousins. *Sundials, The Art and Science of Gnomonics*. Pica Press, 1970.

[28] Winthrop W. Dolan, *A Choice of Sundials*. The Stephen Greene Press, 1975.

[29] Imagine Publishing Inc. *Next Generation*, vol. 2, no. 21, p.52-71. September 1996.

[30] Imagine Publishing Inc. *PC Gamer*, vol. 4, no. 5, p.66-96. May 1997.

[31] IDG Communications. *PC Games*, vol.5, no.6, p.41-57. July/August 1998.

[32] Pioneer Joel. *VRML Form Nightmare*, 1998.

[33] Miguel Gomez. *Simple Intersection Tests for Games*. Gamasutra. October 18, 1999.

[34] Matt Lewis. *Cone Chaser Demo*. http://www.cgrg.ohio-state.edu/~mlewis/VRML/Class/w6/behave/coneChaserDemo.html, 1998.

[35] Thomas C. Hudson, Ming C. Lin, Jonathan Cohen, Stefan Gottschalk, and Dinesh Manocha. *V-Collide: Accelerated Collision Detection for VRML*. Appeared in Proc. of VRML'97, http://www.cs.unc.edu/~geom/collide.html, 1999.

[36] Bryan Stout. *Smart Moves: Intelligent Pathfinding*. Game Developer. July, 1997.

[37] Matt Lewis. *Static Obstacle Avoidance*. http://www.cgrg.ohio-state.edu/~mlewis/VRML/Class/w6/behave/avoidBoxChaserDemo.html, 1999.

[38] Nichimen Graphics. *Mirai*. http://www.nichimen.com/, 2000.

[39] Newtek. *Lightwave 6.0*. http://www.newtek.com/, 2000.

[40] Nichimen Graphics. *Nendo*. http://www.nichimen.com/, 2000.

[41] Takeo Igarashi. *Teddy*. http://www.mtl.t.u-tokyo.ac.jp/~takeo/teddy/teddy.htm, 2000.

[42] Mike Clifton. *sPatch*. http://www.cableone.net/alyson/spatch.html, 2000.

[43] Keith Rule. *Crossroads 3D*. http://www.europa.com/~keithr/Crossroads/index.html, 2000.

[44] Bob Crispen. *VRML Works: File Converters*. http://hiwaay.net/~crispen/vrml/, 2000.

[45] Trapezium. *Vorlon*. http://www.trapezium.com/vorlon.html, 2000.

[46] Trapezium. *Chisel*. http://www.trapezium.com/chisel.html, 2000.

[47] Novafex Software Limited. *Flamingo Optimizer*. http://www.novafex.com/prod_fo.html, 1999.

[48] Cybelius Software. *Cybelius Optimizer*. http://www.cybelius.com/FrmOptim.htm, 1999.

[49] Finney/Thomas. *Calculus, revised printing*. p711-728. Addison-Wesley Publishing Company, July 1991.

[50] Alias/Wavefront. *Maya User's Manual*. 1998.

[51] Guy W. Lecky-Thompson. *Algorithms for an Infinite Universe*. Gamasutra, vol.3, issue 37. September 17, 1999.

[52] Andre LaMothe. *Building Brains into Your Games*. Game Developer. August, 1995.

[53] Carl Muller. B.Sc, *Mathematics in Video Games*. Gamasutra. vol.1, issue 1. June 19, 1997.

[54] Tim Ryan. *The Anatomy of a Design Document, Part 1: Documentation Guidelines for the Game Concept and Proposal*. Gamasutra. October 19, 1999.

[55] Miguel Gomez. *C++ Data Structures for Rigid-Body Physics*. Gamasutra. vol.3, issue 26, July 2, 1999.

[56] Nick Bobick. *Rotating Objects Using Quaternions*. Game Developer. vol.2 issue 26, July 3, 1998.

[57] Andrew S. Glassner. *Graphics Gems*, p.462. Academic Press, 1998.

[58] Gabe Kruger. *Curved Surfaces Using Bezier Patches*. Gamasutra. vol.3, issue 23, June 11, 1999.

[59] Anthony A. Apodaca, Larry Gritz. *Advanced RenderMan, Creating CGI for Motion Pictures*, p.45. Morgan Kaufman, 2000.

[60] Web 3d Consortium. *X3D Documents*. http://www.web3D.org/news/x3d/, 2000.

[61] Sandy Ressler. *Web 3DTtechnologies*. http://web3d.about.com/compute/web3d/, 2000.

[62] 3D EZine. http://www.virtuworlds.com/3DEZine/3dezinef.html, 2000.

[63] The VRML Consortium Incorporated. *Scripting*. http://www.web3d.org/Specifications/VRML97/part1/concepts.html#4.12, 1997.

[64] The VRML Consortium Incorporated. *External Prototype Semantics*. http://www.web3d.org/Specifications/VRML97/part1/concepts.html#4.9, 1997.

[65] Vladimir Bulatov. *Homepage*. http://www.physics.orst.edu/~bulatov/, 2000.

[66] Adobe. *Photoshop*. http://www.adobe.com/products/photoshop/main.html, 2000.

[67] Adobe. *Illustrator*. http://www.adobe.com/products/illustrator/main.html, 2000.

[68] Macromedia. *Freehand*. http://www.macromedia.com/software/freehand/, 2000.

[69] Jasc Software. *PaintShop Pro*. http://www.jasc.com/psp6dl.html, 2000.

[70] Macromedia. *Sound Edit 16*. http://www.macromedia.com/software/sound/ , 2000.

[71] Syntrillium. *Cool Edit*. http://www.syntrillium.com/cooledit/index.html, 2000.

[72] Parallel Graphics. *VRMLpad*. http://vrmlpad.parallelgraphics.com/vrmlpad/site/, 2000.

[73] Free Software Foundation. *Emacs*. http://www.gnu.org/software/emacs/emacs.html, 2000.

[74] ModelWorks Software. *SitePad*. http://www.modelworks.com/, 2000.

[75] David S. Ebert, F. Kenton Musgrave, Darwyn Peachey, Ken Perlin, Steven Worley. *Texturing & Modeling: A Procedural Approach*. AP Professional, September 1994.

[76] Ken Perlin. *Making Noise*. http://www.noisemachine.com/talk1/, 2000.

[77] Parallel Graphics. *Cortona*. http://www.parallelgraphics.com/htm/en/prod/index.html?cort/cort.html, 2000.

[78] Shout Interactive. *Shout3d*. http://www.shout3d.com, 2000.

[79] Gregory Seidman. *Gregory Seidman's VRML2 Widgets*. http://zing.ncsl.nist.gov/~gseidman/vrml/widgets.html, 2000.

[80] Sonic Foundry. *Sound Forge*. http://www.sonicfoundry.com/Products/NewShowProduct.asp?PID=5, 2000.

[81] Sandy Ressler. *General VRML Authoring Tools*. http://web3d.about.com/compute/web3d/msubauth-m01.htm, 2000.

[82] Stepen White. *Dune*. http://dune.sourceforge.net/, 2000.

[83] Craig W. Reynolds. *Steering Behaviors For Autonomous Characters*. Sony Computer Entertainment America. http://www.red.com/cwr, 1999.