



Digital Space 3D Authoring Kit



Copyright © 1997 Digital Space Technologies, Inc. All rights reserved.

Digital Space 3D Authoring Kit, 2.0

P/N 2003-00-0

This document and the software described in it is furnished under license and may be used or copied only in accordance with such license. Except as permitted by such license, the contents of this document may not be disclosed to third parties, copied, or duplicated in any form, in whole or in part, without the prior written permission of Digital Space Technologies, Inc.

This document contains proprietary and confidential information of Digital Space Technologies, Inc. The contents of this document are for informational use only, and the contents are subject to change without notice. Digital Space Technologies, Inc. assumes no responsibility or liability for any errors or inaccuracies that may appear in this book.

Restricted Rights Legend. For defense agencies: Use, reproduction, or disclosure is subject to restrictions set forth in subparagraph (c) (1) (ii) of the Rights in Technical Data and Computer Software clause at DFARS 52.227-7013, and or similar successor clauses in the FAR, or the DOD or NASA FAR Supplement.

Unpublished right reserved under the Copyright Laws of the United States.

Digital Space Talker, Digital Space Traveler, and Digital Space Community Server are trademarks of Digital Space Technologies, Inc.

Windows and Windows NT are trademarks of Microsoft Corporation.

Solaris is a registered trademark of Sun Microsystems, Inc.

UNIX is a registered trademark of X/Open Ltd.

All other products or name brands are trademarks of their respective holders.

Printed in the USA.

Revised by Dominic to reflect updated syntax.

Contents

1 ••• Introduction	5
1.1 What Is the Digital Space 3D Authoring Kit?	5
1.2 Creation Tools	5
1.2.1 RAS Audio Authoring Tool	5
1.2.2 3D Studio MAX Material Library	6
1.3 Traveler Authoring Environment Setup	6
1.3.1 Enabling Single-User Mode in Traveler	6
1.3.2 RAS Setup	9
1.3 3D Studio MAX Material Library	9
2 ••• Quick Start	10
2.1 Familiarity with Digital Space Traveler	10
2.2 Necessary Tools	10
2.3 Supported Data Formats, in Brief	11
2.3.1 Bitmaps	11
2.3.2 Sound	12
2.4 Design and Social Guidelines, in Brief	12
2.5 Performance Guidelines, in Brief	13
2.6 Digital Space Spaces Production Process	14
3 ••• Design and Social Considerations	17
3.1 Overview	17
3.2 Concepts and Terminology	17
3.3 Design for Social Interaction	19
4 ••• Space Design Technical Considerations	23
4.1 Performance Requirements and Optimization Tips	23
4.1.1 Rendering Speed	23
4.1.2 Download Time	25
4.2 Bitmaps in Traveler Spaces	26
4.2.1 Supported File Formats	26
4.2.2 Characteristics of Bitmaps	27
4.3 Color Management	28
4.3.1 How Traveler Creates a Color Palette	29
4.3.1.1 Windows Colors	30
4.3.1.2 Traveler Reserved Colors	31
4.3.1.2 Customizable Colors	32
4.3.2 How 3D and 2D Objects Use Available Colors	32
4.4 Additional Considerations	33

4.4.1 Surface Normals and Back-face Culling	33
4.4.2 Background Images and Boundaries of Space	33
4.4.3 Avoiding Depth Buffer Artifacts	33
4.5 Audio Authoring	33
4.5.1 Definitions	34
4.5.2 Authoring Approaches	34
4.5.3 The Digital Space RAS Audio Authoring Tool	35
4.5.3.1 Using RAS	35
4.5.3.2 RAS Audio Parameter Definition	37
4.5.4 Audio Specifications for Traveler	38
4.5.5 VRML Audio Example	38
5 . . . Additional Resources	40
5.1 Introduction Material	40
5.2 More Information on VRML	40
5.3 Standards	41
5.4 More Readings	42
A . . . VRML Specifications	43
A.1 VRML Standards	43
A.2 Digital Space VRML Extensions	43
B . . . Converting DSS Spaces to VRML	55
B.1 Necessary Tools	55
B.2 Conversion Process	55
B.2.1 Converting Bitmap Files to PNG	56
B.2.2 Converting .3ds Files to VRML	58
B.2.3 Converting DSS File to VRML	61
B.3 DSS Specification	72
B.3.1 Overview	72
B.3.2 File Format	72
B.3.3 Data	72
B.3.4 .dss File Extension	74
B.3.5 Header	74
B.3.6 DSS Keywords	75
B.3.7 DSS Example	83
C . . . Appendix	
C.1 VRML Quick Reference	86
C.2 DSS to VRML Quick Reference	95

1••• Introduction

1.1 What Is the Digital Space 3D Authoring Kit?

The Digital Space 3D Authoring Kit 2.0 includes documentation, references, and tools to aid 3D authoring designers in easily developing Digital Space Traveler VRML spaces. It enables customers to create, test, and bring up Digital Space spaces for either their own use (e.g., server customers) or for clients that they are producing sites for (e.g., Web site production houses). Currently, this kit encompasses VRML 3D space authoring only; avatar authoring guidelines are discussed in the *Digital Space 3D Avatar Authoring Kit*.

This kit supersedes the Digital Space Authoring Kit 1.0, which was a toolkit for creating spaces using DSS and 3D Studio data formats. The 2.0 version includes instruction for creating VRML spaces from scratch, in addition to guidance for converting DSS spaces to VRML file formats.

Note

This document is not to be considered a VRML tutorial. A working knowledge of the VRML specification and a familiarity of referenced tools is crucial to understanding many concepts covered in this authoring kit.

1.2 Creation Tools

The Digital Space 3D Authoring Kit 2.0 includes the following tools and data:

- *ras.exe* - Digital Space's RAS (Random Audio Sequencer) tool, version 2.0.
- *Dstrav.mat* - A 3D Studio MAX® material library file, containing 24 base Digital Space Traveler colors.

The authoring kit components are described in greater detail in the following subsections.

1.2.1 RAS Audio Authoring Tool

The audio authoring tool included in the kit is called *RAS*, for Random Audio Sequencer. The RAS tool is an Digital Space patented 8-channel randomized audio tool used for creating ambient audio and local sound effects for Traveler spaces. Through the RAS interface, designers can load *.wav and *.olw files into an audio channel parameter list, define their time boundaries for randomized playback as either "fixed" or "min/max", and then see how the files sound as ambient audio in a Traveler space.

1.2.2 3D Studio MAX Material Library

The material library file, *Dstrav.mat*, contains Digital Space Traveler's base 24 shaded colors. Traveler fills 192 entries of 256 color palette with these 24 base colors that are ramped from light to dark with eight different intensities. Refer to section 4.2.2 for details on how Traveler use colors.

1.3 Traveler Authoring Environment Setup

This section describes the setup procedures of a basic Traveler authoring environment, such as enabling the single-user mode of Traveler and setting up the tools included in this kit.

1.3.1 Enabling Single-User Mode in Traveler

With Digital Space Traveler, users connected through the Internet interact with each other by talking and moving throughout a Traveler space. The Traveler spaces are normally connected to an Digital Space Community Server that enables the user to see a Traveler 3D space and seamlessly interact with other people, or avatars. This networked multi-user interactive environment seems ideal in terms of the users needs. For a space designer, however, such an interactive environment is distracting as users are continually entering, exiting, and moving around in the space.

For designers wanting to create or modify the artwork of Traveler spaces, a "single-user mode" of Traveler can be enabled. With single-user mode, a designer does *not* have to see, hear, or talk to others in Traveler spaces, but they can still move around, view, and hyperlink to other local spaces (assuming that the URLs are modified according to your connection mode). Designers can edit, and view spaces privately that exist either locally on their hard drives or on servers behind the firewall without the worry of others entering a space that they are working on. Single-user mode is not a special version of Digital Space Traveler; it is a mode that can be enabled when designing spaces. Designers can just as easily switch back to interactive mode when they want to use Traveler in its full networked multi-user capacity. Enable single-user mode in Traveler as follows:

1. *Set Traveler to launch without a Web browser.*

Insert `BypassSelectAnAvatar=-1` at the end of the [Browser] section of the `dstrav.ini` file (usually located in the `c:\Windows` directory). For example:

```
[Browser]
...
BypassSelectAnAvatar=-1
```

This enables Traveler to launch without launching your default Web browser first.

2. *Disconnect Traveler from the Digital Space Community Server.*

To temporarily disconnect a Traveler space from the community server, map the server name to a `NO_CONNECT` alias, as follows.

- a. Insert an `[Indirect]` heading at the end of the `dstrav.ini` file.
- b. Add `All_Servers=NO_CONNECT` after the new heading, as follows:

```
[ Indirect ]
All_Servers=NO_CONNECT
```
- c. If connected to the Internet, save and exit out of the `dstrav.ini` file, then go to step 4. If *not* currently connected to the Internet, go to step 3.

3. *Turn caching off.*

If connected to the Internet, skip to step 4.

If not connected to the Internet, turn caching off, as follows:

- a. Insert `DefaultFrequency=-3` under the `[cache]` heading in the `dstrav.ini` file, as follows:

```
[ Cache ]
DefaultFrequency=-3
```

This tells Traveler to never check the server for new data.

- b. Save and exit out of the `dstrav.ini` file, then go to step 4.

4. *Modify URLs in the *.dsv or *.wrl files, as needed.*

- a. Use either the HTTP (`http://<host>/<path>`) or file (`file://<host>/<path>`) URL schemes in the VRML files to specify the location of resource data.

For example:

<http://anyserver.com/places2a/entrance.dsv>

or

`file://c:/any_path/entrance.dsv`

- b. It is highly recommended that you use the same directory structure for your data files as the final directory tree on the server. If you

do, you can use relative URLs and minimize the amount of necessary changes to the VRML files. The only required change in URLs would then be the home address of the file which is specified in the `url` field of the `Dspace_Server` node.

For example, if your final data files are organized on an HTTP server with a directory tree like the following:

```
http://anyserver.com/places2a/maps/*.png
http://anyserver.com/places2a/props/*.wrl
http://anyserver.com/places2a/sprites/*.png
http://anyserver.com/places2a/texmaps/*.png
http://anyserver.com/places2a/wavs/*.olw
```

the working directory structure of your project on the local C drive would look similar to the following:

```
c:\any_path\maps/*.png
c:\any_path\props/*.wrl
c:\any_path\sprites/*.png
c:\any_path\texmaps/*.png
c:\any_path\wavs/*.olw
```

Note that once you have finished Digital Space Traveler space design efforts, and it is ready to put up on a server, some URLs will need to be re-edited (probably only the home address of the main file).

For example, an authoring setup like the following:

```
#VRML V1.0 ascii
Separator {
  Dspace_Server {
    fields [ SFString server,
             SFString url ]
    server "anyserver.com"
    url "file:///c:/any_path/entrance.dsv" }
  # rest of the file}
```

would need to be changed to reflect the http server URL instead of the local file URL address, as shown in the following:

```
#VRML V1.0 ascii
Separator {
  Dspace_Server {
    fields [ SFString server,
             SFString url ]
    server "anyserver.com"
    url "http://anyserver.com/entrance.dsv" }
  # rest of the file}
```

1.3.2 RAS Setup

Save the `ras.exe` file from the Authoring Kit package to any directory that you wish, then create a shortcut to it and place it on your desktop. Launch RAS by double-clicking on the shortcut icon, and the RAS interface is displayed.

1.3.3 3D Studio MAX Material Library Setup

The default location for the 3D Studio MAX material library files is the `maps` sub-directory in the main application directory (e.g., `c:\3dsmax\maps`). Copy the `Dstrav.mat` file into this directory. From the 3D Studio MAX Material Editor, open the library and pick the material that you need.

2 • • • Quick Start

This section has been developed for 3D designers who already have significant experience in 3D authoring, including an understanding of VRML concepts and benefits.

It covers the following areas:

- Information on where to go for further information on the “how-tos” of Traveler, from a client perspective.
- The necessary tools required for designing custom Traveler VRML spaces.
- Definition of supported data formats, in brief, for Traveler VRML spaces.
- Design and social considerations, in brief, to be carefully planned for in the design of a Traveler VRML space.
- Performance guidelines, in brief, to be carefully planned for in the design of a Traveler VRML space, and to ensure that your space works smoothly with the Traveler client networked to an Digital Space Community Server.
- The Traveler VRML space production process, in brief.
- Digital Space Traveler 2.0 VRML quick reference table.

2.1 Familiarity with Digital Space Traveler

By the very fact that you are reading this document, we can assume that you are very familiar with Digital Space Traveler and what it can do. For more information on Digital Space Traveler from a user perspective, refer to the *Digital Space Traveler User Guide* available at :

<http://traveler.digitalspace.com/tech/trav/ug/travug20.html>

2.2 Necessary Tools

To create Digital Space Traveler spaces, you need to have the following software installed:

- A 3D authoring tool that supports VRML 1.0 output (e.g., *Kinetix 3D*)

Studio MAX or ParaGraph Virtual Home Space Builder)

- An image editor that supports PNG, BMP, and JPEG file formats (e.g., *JASC Paint Shop Pro*)
- A text editor
- Digital Space Traveler v2.0 (or later) client application

2.3 Supported Data Formats, in Brief

A VRML file is a textual description of your space, including 3D objects, their properties, and transformations. It also supports the inline inclusion of encoded data, such as bitmaps and sound files. The following sections describe these data formats in more detail.

2.3.1 Bitmaps

Traveler 2.0 (or later) supports PNG (Portable Network Graphics), JPEG, and Windows BMP bitmap formats. PNG is the most flexible among the supported formats. This format supports indexed-color, grayscale, and true color images. It also supports a full alpha channel as well as transparent-color specifications. PNG uses a lossless but efficient compression scheme.

The JPEG (Joint Photographic Experts Group) file format offers a high rate of compression, and it is used particularly for continuous-tone images. Compactness of this format helps the transmission of images over the Internet. It is a lossy compression technique because it discards image information as it compresses the file. JPEG file format can only be used to store 24-bit images with no transparency information.

Windows BMP is a general purpose format for imaging applications operating in Windows environments, but which is also supported by many non-Windows and non-PC applications. BMP files can either be stored uncompressed or compressed, using a type of run-length encoding (RLE). It can accommodate images up to 24-bit color.

Table 1 specifies the suggested format for each group of bitmaps in the Digital Space spaces, background images, sprites, and texture maps. If you use JPEG or BMP formats, refer to section 4.2.1 to better understand the implications of using these formats on your spaces.

Bitmap Type	File Format	Subtype	Color Depth	Transparency	Pixel Dimensions
Background Images	PNG	Non-interlaced	8-bit	None	2 ⁿ value for width and height
Sprites				Use transparent-color specification, if needed	
Texture Maps					

2.3.2 Sound

Digital Space Traveler supports the standard *.wav* file format, as well as the *.olw* file format, an Digital Space proprietary compression sound file format. By using the *.olw* format, sound files in Traveler are much smaller, which considerably reduces the download time during playback. Refer to section 4.5 for more details.

2.4 Design and Social Guidelines, in Brief

- Following are some considerations for designers to carefully evaluate as they plan the development of their Traveler space(s).
- The main design objective of spaces is to support communication and social interaction among participants in a multi-user environment.
- The overall scale of a space should be optimized for approximately 10 to 15 avatars (an average avatar is about half a meter tall).
- A space should have several stages or meeting areas with one or two relatively recognizable main stages and several secondary stages.
- The overall size of a space should be large enough to accommodate four or five “private” conversations of three or four people, but small enough so that users can easily view and quickly navigate to other conversations.
- The entrance area or landing points should be away from staging areas so arriving avatars do not unduly bother those already in conversations.
- The entrance area should be located where arriving avatars can oversee some of the activities in the space and thereby are encouraged to investigate the space and avatars already present.

- Pathways should be designed to facilitate comfortable and convenient movement through space, interaction among avatars, and access to key places and objects.
- Portals to other spaces should be recognizable as a portal from a distance, be multi-dimensional for easy access and viewing from different angles, and be easily learned.

2.5 Performance Guidelines, in Brief

Following are some performance guidelines to ensure that a designer's Traveler space(s) runs as efficiently and smoothly as possible. For more details, refer to section 4.

Rendering Speed

The rendering speed of an Digital Space Traveler space, occupied by at least a few avatars, should be kept above 12 FPS (frames per second) in a 320x240 window size on a 100 MHz Pentium computer to ensure that avatars have smooth lip synchronization and comfortable movement through the space.

Some ways to improve the rendering speed of your scene are:

- Total number of 3D triangular polygons in a space should not exceed 1500.
- Flat shading should be used instead of smooth shading (as much as possible).
- Avoid closed spaces or the use of large objects and surfaces that cover a big portion of the viewport.
- For large surfaces, like floors and walls, use self-illuminant materials.
- Texture mapping should be used sparingly. If you need to use them, use small texture maps on small areas.
- Sprites should be used instead of texture maps, wherever possible.
- Lights should be used sparingly. Try not to specify more than four light sources in a space.

Download Time

Keep total size of all files used in the space below 500 KB to ensure that initial downloading of the space does not exceed 5 minutes over a 28.8 Kbps connection line.

To make your files smaller:

- Keep pixel dimensions of your bitmap files as small as possible.
- Use 8-bit bitmaps instead of 24-bit bitmaps.
- Generally, PNG file format is the preferred file format for bitmaps because it is a lossless but efficient compression scheme, and its flexible.
- If BMP file format is used, compress the files using RLE method.
- Compress *.wav sound files to *.olw format.
- In *.wrl or *.dsv files, use VRML geometric primitives, like Sphere, Cone, and Cylinder, instead of IndexedFaceSet, only if your objects have more or equal number of polygons than the default number set by Traveler.
- Traveler automatically generates the surface normals, so do not include them in *.wrl or *.dsv files, unless you need to manipulate them manually.
- Use instancing for the data that is used more than once in a space.

2.6 Digital Space Spaces Production Process

This section describes the primary steps to create Digital Space VRML spaces, in brief. For more details, refer to section 4.

1. Conceptualize and design your spaces according to the design, social, and technical guidelines explained in this document, including VRML implementation in Traveler.
2. Model your space with a 3D modeling tool of your choice.
3. Save your space as a standard VRML 1.0 file (*.wrl).
4. As the minimum requirement, you can turn your standard VRML 1.0 space into an Digital Space Traveler space by just adding four nodes to your space. A suggested procedure is as follows:



- a. Create a new empty file in a text editor, and save it with an *.dsv* file extension.
- b. Add the required VRML header to your file: `#VRML V1.0 ascii`
- c. Create the root `Separator` node, and then add the following children nodes to it:
 - Add the `Digital Space_Server` node to specify the name of the Digital Space Access Server which the space is connected to and url of the **.dsv* file.
 - Add the `WorldInfo` node and specify the name of the space in the `title` field.
 - Add the `Dspace_MaxAvatars` node to specify maximum number of avatars allowed in an instance of the space.
 - Add the `Dspace_MaxInstances` node to specify maximum number of instances or clones of the space that can be created in your Digital Space community.
- d. Add a `WWWInline` node as a child node to the root separator node and specify the URL of your VRML file, saved in step 3, in the `name` field.

You now have a fully functional Digital Space Traveler 2.0 space. The following recommended steps are optional.

5. Add the optional Digital Space features to enhance your space both socially and asthetically.

Optional Features	Implementation
Audio	Add one or more DSpace_Sound nodes to define the sound sources for ambient audio and/or local sounds.
Background	Add one or more DSpace_Horizon nodes to specify a background texture at infinite distance. Add the Background node to specify color backdrops that simulate ground and sky and a tiled texture.
Ambient Lighting	Add an Environment node to define global ambient lighting.
Space Bounding Box	Add the DSpace_CubeBoundary node to define a box that bounds the avatars' movements in space.
Entry to Space	Add one or more DSpace_EntryPoint nodes to define attributes of the avatars' entries into the space, such as landing position and orientation.
Spinning Objects	Use the DSpace_Spin node to specify a constant 3D spinning for certain objects about an arbitrary axis through the origin for subsequent objects.
Sprites	Use the DSpace_Sprite node to create a 2D bitmap object that always faces the viewer.
Hyperlinking Spaces	You can link your spaces to other spaces or Web pages by using the WWWAnchor node. This way you can make any 3D or 2D in your space object a portal or a link. To make the hyperlinked space a user space, add the DSpace_UserPortal node. It sets the subsequent WWWAnchor node to be a user portal.

6. Fine tune and test your files.

3 • • • Design and Social Considerations

3.1 Overview

In general, a community consists of people interacting and communicating with each other in several possible locations within the boundaries of that community. In Digital Space virtual communities, people are represented as 3D head avatars and community locations are represented as bounded 3D spaces. The primary channel of communication in these virtual communities is through real-time multi-point voice, avatar facial movements and expressions, and an avatar's abilities to navigate in the environment to communicate with others.

The primary function of an Digital Space space design is to create an optimal environment for communication and socialization. The space can create the mood, supply the content, enhance and structure the social group dynamics and in general create the structure where socialization happens.

A Digital Space community is a collection of spaces that utilize the same Digital Space community server. How the spaces relate to each other and are connected or hyperlinked together define the community structure. A Digital Space community can be considered a neighborhood or an affinity group.

Space design (in terms of this document) is about:

1. creating individual environments that enable social life in a variety of flavors, and
2. structuring and interconnecting some number of these separate spaces into a coherent form that defines a community.

3.2 Concepts and Terminology

Digital Space Traveler is a multi-participant, voice enabled 3D (VRML) browser that allows people to communicate and socialize with each other in virtual communities. Each person who enters the virtual communities via Traveler is embodied as an avatar. An *avatar* is a 3D object represented as a human or animal-like head (but also could be a fish, guitar, eyes, mouth, etc.) that moves, speaks (lip syncs with person's voice), and shows emotion in a virtual space. The goal is that the real person connected via Traveler and his/her avatar representation are so connected that it is hard to define a separation of the two; hence this document often uses the terms "avatar" or "person" (Traveler user) interchangeably to describe them.

Spaces are bounded 3D worlds where avatars or people meet and socialize. Within a space, there are areas called *stages*, which are loosely defined areas where avatars meet and converse. They are not distinct areas; they are merely the location within a space where avatars congregate in discussion. Depending on how many avatars are in a space, there could be two or three (or more) stage areas in a single space. Because a finite number of avatars are allowed in a space at any given time (currently 10 to 12 avatars), a space has a scale that best accommodates the social group dynamics for this number of avatars. This makes the typical size of a space somewhere between a medium size room to a large public plaza. Spaces can be designed for different functions or moods, but in general, their main purpose is to create the best environment for avatar/person communication and socialization.

The mechanism for traveling from one space to another is called a portal. *Portals* are represented as 3D or 2D objects in a space that when selected (or collided with), take the avatar via a hyperlink to another specified space. Portals use the standard Web-based URL mechanism to define the destination that the avatar will travel to. *Links* function similarly to portals in that they use the same URL hyperlinking mechanism, and they are represented as 3D or 2D objects in a space. Links are activated by being selected (or collided with). Links, however, do not take an avatar to a new space. Instead they bring up (in the standard web browser window) a Web object specified by the URL, which is typically another HTML page.

A *community* is a set of interconnected spaces that use the same Digital Space Name Server (component of the Digital Space Community Server). The connectivity between spaces is achieved through portals. An avatar/person can use community based tools to find out where his or her neighbors are, which spaces are the most popular, and travel to a space where their friend is. These tools work only within a community scope, so if a friend is currently in a different community, their name would not show up using these tools. A community is sponsored and maintained by a particular site owner who produced (or commissions the production of) all the spaces, including the specific portal hyperlinks between community spaces and links to different HTML pages.

A community has one space called the *entrance space*, which is considered the entrance or main lobby to the community. Graphically, the entrance space is the space that best represents the community. An entrance space typically contains many portals to all other spaces within the community, and has links to relevant community information HTML Web pages. The “other” spaces (non-entrance space) should always have a portal that hyperlinks back to the entrance space, the central hub of the community.

Depending on how the entrance space is designed, often the entrance space also includes portals to access the entrance spaces of different communities. If a portal to a specific community is not available, an avatar can also travel to

any community or space by typing in the URL of that space.
When designing spaces for a community, there are three main categories of spaces to consider:

1. Entrance space - initial area of gathering and socialization.
2. Chat spaces - for more specific types of conversation and socialization.
3. User spaces - user-defined chat spaces for a semi-private conversation.
The user spaces can be password protected for private meetings.

Generally, the level of personalization increases as an avatar travels from category 1 to 3. In addition, some physical aspects of the space and its functional elements, such as scale and circulation system of the space, change accordingly.

3.3 Design for Social Interaction

There are mainly three interrelated design concerns: social/functional issues, technical issues, and aesthetics. In this section we discuss the first one.

Multi-Participant Virtual Environments

When designing spaces for Traveler, you are not designing one person virtual worlds. Since the environments are occupied/used by multiple participants, interacting with each other (mainly through conversing), it is important to evaluate these different worlds carefully. Creating multi-participant versus single person virtual environments imposes a whole different set of design issues, with the key design objective being the support of communication among participants.

One of the main prerequisites for one's active participation in a virtual community and engagement in social interaction with others is that person's experience of presence in a "non-real" world. With this, one of the core goals of designing the virtual environments is to create a sense of place in which this feeling of presence is supported and enhanced.

The user's sense of being in the virtual world can be enhanced by the environment's acknowledgment of this being. The acknowledgment takes the form of reacting to the participant's movement and behaviors with its own specific types of behaviors. For example, an object's sound, collision sound, and portals and links that transport or link the user to other virtual entities are all examples of the environment's acknowledgment of somebody's being.

This acknowledgment is reinforced by certain user interface elements in the Traveler.

Among other general design issues, the following considerations are most significant:

- *Psychological Comfort* - Creating conditions for mental ease with appropriate forms, shapes, colors, symbols, lighting, and sound.
- *Convenience* - Make it easy to access spaces, people, objects, and information
- *Flexibility* - Availability of choice, variety, and future expansions/ extensions.
- *Maintenance* - Keep the virtual environment clean and manageable.
- *Personalization/Privacy* - Availability of private or semi-private areas through proper organization of the community, and congregating areas within spaces (stages).
- *Copyrights* - Respect intellectual properties of others when using prebuilt objects.

Spatial Definition

The size of avatars affects the volume of space they require for movement and interaction with each other. Avatar dimensions also affect the dimensions and proportions of spatial elements, the height and distance of things they must reach, and various objects used in a place. With all this in mind, the bounding box of a typical avatar is 0.5x0.5x0.5 meter cube.

The overall scale of a space should be optimized for approximately 10 to 15 avatars and the overall size of a space should be large enough to accommodate 4 or 5 “private” conversations of 3 to 4 people within the space. On the other hand, the space should still be small enough so that users can easily view and quickly navigate to other conversations or areas of the space. “Private” in terms of a single space means far enough away so as to not be heard by the next closest conversation or stage.

Movement through Space

Traveler users experience a space through moving in it. Every space has an explicate or implicitly defined circulation system to accommodate the goal of comfortable and convenient interaction among avatars.

Different components of the circulation system are the approach to a space, the entrance, and sub-spaces, such as stages, meeting, and conversation areas, portal/link areas (we can consider portals as exits from one space to another), and the pathways that connect all these elements. Every part of this system

must be designed in connection with all other related parameters. (Refer to the section on “Stages—Spaces within Spaces” on page 35 for better understanding of “stages”).

Approach

The approach to a place is specified by one or more `Dspace_EntryPoint` node(s). It can be a downward fly-through from the top or other sides on a spiral path. As the designer, you can of course design the approach any which way you want; the point being made here is simply that approach is a design consideration that can easily be overlooked if not specifically thought through.

Entrance

The entrance area is where avatars start out when they enter the space. They should be away from staging areas (informal discussion areas within a space) so new avatars do not unduly bother those already in conversation.

The entrances should be located where new arrivals can see the stages where conversations are occurring, allowing, these avatars to oversee some of the activities in a space and thereby encouraging them to investigate the place and its avatars. The entry points of a space can be randomized by use of multiple `Dspace_EntryPoint` nodes.

Pathways

Natural pathways are preferred since by their design encourage interaction.

Stages—Spaces within Spaces

Stages are loosely defined areas where avatars meet and converse.

A space should have several stages, or meeting areas, with one or two relatively recognizable main stages and, several secondary stages. The main stage allows users coming into a space for the first time to easily recognize a place to go to converse with others. The secondary stages allow people to break off from a large group and start a sub or private conversation. This can be accomplished by a change of levels or by defining an implicit enclosure.

The form and scale of these sub-spaces must accommodate the level of traffic and movement and flow of avatars (and information).

Stages usually have visual dominance within the space, and this dominance should be achieved without losing the flow of movement. The spatial and visual continuity within the whole space must be preserved.

Connecting Places to Form a Community

The way that communities are usually organized is that in each community there is an entrance space where the avatars initially gather and socialize. The entrance space is then connected to other spaces that have different contents/context/design but still within the general theme of the community. People with similar interests gather in these spaces for more specific type of conversation and socialization. If any of the participants within a space wants to have a more private conversation with others, she or he can create a temporary *user space* (of any name that the person desires).

Participants can create as many user spaces as they want. Although, after all avatars have left a user space, it is deleted.

Portals are the means of connecting the spaces together, using 3D or 2D objects that can be “portalized” by assigning a URL of another space to it.

Following are some important properties of objects that are used to represent portals:

- Recognizable as a portal from a distance.
- Multi-dimensional, easily accessed and viewed from many angles.
- Easily learned.

Portals could be considered as space exits or departure areas (and entryways to the space that the portal is hyperlinked to). Therefore, special attention to their positions, pathways leading to them, and their relationship to other spatial elements is needed.

Hyperlinks to Web Based Content

You can enrich a virtual community by proper integration of the 3D worlds with the rest of your Web content. This can be done by adding controls within spaces to access any type of information available on the Internet, and by adding hyperlinks on your Web pages to Traveler spaces. The hyperlinked objects in a space automatically become links if they point to any Web object other than another Traveler space (the latter defining a portal, not a link).

4 • • • Space Design

Technical Considerations

4.1 Performance Requirements and Optimization Tips

This section provides more details on some aspects of the performance guidelines, briefly discussed in section 2.5.

4.1.1 Rendering Speed

The rendering speed of an Digital Space Traveler space is the key element for the performance requirement. For a typical space, with at least a few avatars in it, the rendering speed should be kept above 12 FPS (frames per second) in a 320x240 window size on a 100 MHz Pentium computer to ensure that avatars have smooth lip synchronization and comfortable movement through the space.

In general, the combination of the following three parameters directly affects the frame rate.

- total number of polygons used in the space
- object size and coverage of view frame
- shading model used to render surfaces, including surface properties

Balanced distribution of these parameters through the entire space produces a consistent frame rate as the avatars move around the space and speak. There are of course other factors that affect rendering speed of a space, but their impact is usually not as significant as the above mentioned parameters.

An important thing to keep in mind is that the number of avatars present in a space, and the fact that they may all be within close range of one another at times, that there will be a significant impact on rendering speed. Therefore, when testing a space that you design, be sure to populate it with at least a few avatars, to get a good feel for the rendering impact.

Polygon Count

A space's geometric complexity should normally be limited to about 1200 to 1500 3D polygons (single-sided triangles). This refers to the total number of polygons used in individual 3D data files, including portals, links, and spinning objects, and does not include avatars.

VRML Geometric Primitives

Use of geometry nodes, such as Cone, Cylinder, Cube, and Sphere, is usually more compact than describing the same object with the IndexedFaceSet node. It also makes the files smaller. However, you need to take this into consideration that in order to display and render geometric primitives, Traveler internally converts them into polygonal meshes. It uses 64 triangular polygons for a cone, 12 polygons for a cube, 96 polygons for a cylinder, and 512 polygons for a sphere.

In many cases, specially when you need to use spherical objects in your space, it is better that you create a low polygon version of the primitive and save/export it as an IndexedFaceSet node so that you stay within your polygon budget.

Shading Models and Surface Characteristics

The choice of polygon shading (shading model) and the properties of surfaces, such as self-illuminancy (unlit flat) and texture mapping, also affects rendering speed, so planning out your shading is important. The following are the available polygon shading options listed from the most to the least computationally expensive, in terms of how it affects your rendering speed:

- *Texture-mapped* - currently slow, should be used sparingly. Use it on small surfaces, if needed. Or, use sprites, if possible.
- *Gouraud (or smooth) shaded polygons* - second slowest method but good for curved surfaces.
- *Flat (or facet, constant) shaded polygons* - fastest shading method that shows up lighting.
- *Self-illuminated (or emissive, unlit flat) polygons* - slightly faster than flat but has no 3D shading.
- *2D rendering* - used for horizon and sky tile bitmaps.

Most current spaces are primarily flat shaded with lots of self-illuminated and 2D rendered surfaces.

Object Size and Coverage

Many small objects that are well dispersed throughout the space are much less computationally expensive than large objects that take up a lot of the view frame. Using many well-dispersed small objects also aids in 3D depthcueing.

Large surfaces, such as walls that take up most of the view frame, significantly reduce rendering speed and should be avoided, or else you should use the fastest rendering technique, which is typically self-illuminated shading.

4.1.2 Download Time

The total size of all files used in a space should be kept below 500 KB. The main reason behind this requirement is to ensure that the downloading of the space does not exceed 5 minutes for the users connected to your Digital Space community via a modem line.

The performance guidelines in section 2.5 are self-explanatory; we elaborate only on a few of those items in this section.

VRML Geometric Primitives

Refer to “Polygon Count” on page 38 for more information on this subject.

Surface Normals

The surface normals are vectors or straight lines with a specific direction, located on the vertices of each polygon. They are used to define the orientation of a surface for back-face culling and calculation of surface shading. Traveler is able to calculate the normals without any explicit definition of them in the VRML files. It uses face normals for flat shading and vertex normals for smooth shading. Therefore, by eliminating all explicit references to surface normals (e.g., the `normalIndex` field in the `IndexedFaceSet` node), or by not having the VRML exporter to automatically generate them at all, you can make your files more compact. One of the rare occasions that you might need to keep the normal definitions in your file is if you plan to manipulate them manually to create some special shading effects.

Redundant Data in VRML Files

You can make your VRML files more compact (and faster to download) by eliminating the unnecessary information in them.

White Space

In the VRML files, the general rule with white space is that you need to keep a balance between compactness and readability of your files. If you or anyone else needs to manually edit or post-process the files, it is important that you keep the indentations in the code, at least, even if you are eliminating extra blank lines or tabs and spaces at the end of the lines.

Floating Point Numbers

In most cases, 3 decimal places in floating point numbers are sufficient for accurate description of your scene with a typical scale of a Traveler space. So you can round the numbers to 3 or fewer decimal places. Some VRML exporters, like the 3D Studio MAX plug-in, allows users to specify the number of decimal points during the conversion process. Table 3, “Traveler 2.0 VRML Quick Reference,” on page 17

When using fewer decimal points, it is important that you always pre-test your entire scene or sample pieces of it with different settings. The reason for this is that, in some cases, the 3D objects start falling apart or lose their accuracy after passing a threshold.

Default Field Values

In the VRML files, if a value for the field is not specified, the default value for that field is used. So you can eliminate the fields with default values to make your files smaller. Refer to Table 3, “Traveler 2.0 VRML Quick Reference,” on page 17 for detailed definition on default values of the fields in every available node.

4.2 Bitmaps in Traveler Spaces

This section discusses the various bitmap file formats that are supported by Traveler 2.0, and the main features of each of them.

4.2.1 Supported File Formats

Traveler 2.0 supports PNG, JPEG, and BMP formats to display background images, sprites, and texture maps in the Digital Space spaces.

PNG

PNG (Portable Network Graphics) is the most flexible of the supported file formats. It is an extensible file format for the lossless, portable, well-compressed, and efficient storage of bitmap images; it is designed to work well in online viewing applications, such as World Wide Web. PNG provides a patent-free replacement for GIF. Indexed-color, grayscale, and true color images are supported, plus an optional alpha channel as well as transparent-color specifications.

JPEG (JPG)

The JPEG file format, from Joint Photographic Experts Group, offers high rate of compression, used particularly for continuous-tone images. Compactness of this format helps the transmission of images over Internet. JPEG works by removing data that is redundant or data whose removal is almost imperceptible to the human eye. It is a lossy compression technique because it discards image information as it compresses the file. Keep in mind, however, that by increasing the compression rate, the image quality decreases.

JPEG file format can be used to store only 24-bit images with no transparency information.

BMP

Windows BMP format is a general purpose format for imaging applications, operating in Windows environment; BMP is also supported by many non-Windows and non-PC applications as well. BMP files can be stored either uncompressed or compressed, using a type of run-length encoding (RLE). It can accommodate images up to 24-bit color.

RLE is a common compression algorithm used by several bitmap formats, including BMP, to reduce the amount of redundant graphics data. Although RLE is lossless, it is not considered a superior compression method. Pixel depth in BMP files can be 1, 4, 8, or 24 bits (4-bit and 8-bit images can use RLE). Traveler supports only 8-bit format for background images and texture maps; sprite bitmaps can be either 8-bit or 24-bit.

4.2.2 Characteristics of Bitmaps

Among various characteristics of the bitmap images, pixel dimension, color resolution or depth, and transparency are the most important in terms of working with Traveler. There are certain considerations, requirements, and constraints to consider when using different file formats for each group of bitmaps.

Pixel Dimension

Every bitmap image contains a fixed number of pixels, measured in pixel height and pixel width; this is called the pixel dimension of an image. For background images and texture maps, Traveler works most efficiently with bitmaps that have widths and heights equal to a 2^n value (e.g., 8 pixels by 8 pixels or 64 pixels by 128 pixels)—this constraint does not apply to sprites, however. For this reason, Traveler automatically resamples all PNG and JPEG bitmaps which do not meet the 2^n value criteria so that their widths and heights are adjusted to the closest 2^n value. BMP files are required to have

widths and heights of 2^n value, otherwise the images will not be rendered properly.

It is recommended that you follow the above criteria for all formats to avoid unpredictable or poor quality resampling.

Color Resolution

Color resolution, also called color depth, pixel depth, bit depth, or bit resolution, refers to how much color information is available for each pixel in an image. Greater pixel depth means more available colors and more accurate color representation in the digital image. A 1-bit pixel can be one of two colors, a 4-bit pixel can be one of 16 colors, and so on. The most commonly used pixel depths are 1, 2, 4, 8, 15, 16, 24, and 32 bits.

Traveler always uses 256-color model to render the spaces, regardless of current Windows display mode. This means that even if you use a higher color resolution than 8-bit for your images, Traveler reduces the number of colors to 256 or less. Refer to section 4.3 for more information on this subject.

Some image editing applications and utilities, such as Adobe Photoshop or DeBabelizer, use pretty good color reduction algorithms. We recommend that you use these application or similar ones to reduce the number of colors in your bitmap images to the fewest possible number, rather than letting Traveler do it automatically, resulting in a visually not-so-efficient or desirable color reduction. Once you do this, you can also save your images in the most appropriate file format in order to make the files as small as possible; 8-bit PNG is the most commonly used format.

Transparency

Among the supported bitmap formats, PNG is virtually the only format that can hold transparency information for texture maps and sprites (transparency in background images is not supported). Although you can specify the black pixels as transparent areas in BMP sprites, it is not a standard and flexible way of authoring bitmaps that need to contain full or semi-transparent areas.

4.3 Color Management

This section describes the way colors are handled by Traveler. It also provides some tips for the use of colors in the spaces.

One of the new features in Traveler 2.0 is that it can run in various Windows display modes (i.e., 16, 24, and 32-bit modes in addition to 256-color mode). This feature is quite useful when you are testing your spaces in Traveler and need to run other applications, like image editors, that work best with truecolor

displays at the same time.

However, Traveler always uses only 256 colors internally, regardless of the selected display mode. It algorithmically creates a color palette for a space and then the system finds the best fit colors for all the visual elements in the space from the available colors in the palette.

Of course, you can skip these technical details and perform the ultimate test. Simply design your space with colors you want, open it in Traveler, see how it looks, and then fine-tune the colors as needed.

4.3.1 How Traveler Creates a Color Palette

Traveler uses Reality Lab as its rendering engine. To keep the rendering speed high, it uses the “ramp” (or palette) color model (versus RGB model).

This means that Traveler creates a 256-color palette for each space and uses those colors to render the entire space, including 3D objects, bitmaps, and avatars.

There are mainly three segments in Traveler palettes that constitute the total available 256 colors:

1. Windows system colors (20 palette entries)
2. Traveler reserved colors (192 palette entries)
3. Customizable colors (44 palette entries)

4.3.1.1 Windows Colors














Windows uses 20 entries of the palette for standard system colors. The RGB value of 16 of them is fixed. Although the other four have default values which are customizable by the system. For example, when you change the color scheme of your desktop, new RGB values are assigned to these four entries.

Although Traveler cannot change these 20 entries in the palette, you can use the fixed portion of it (16 colors) in your design. They are highly saturated colors. The other four entries vary from one computer to another, so you must not use them at all.

Note

Note that in VRML, the RGB components of colors are specified by using a range between 0 to 1, whereas we use the 0 to 255 convention in this section.

The Following table lists the RGB values of the Windows colors:

No.	Palette Index	R	G	B	Note	
1	0	0	0	0	Black	
2	1	128	0	0	Dark Red	
3	2	0	128	0	Dark Green	
4	3	128	128	0	Dark Yellow	
5	4	0	0	128	Dark Blue	
6	5	128	0	128	Dark Magenta	
7	6	0	128	128	Dark Cyan	
8	7	192	192	192	Light Gray	
9	8	192	220	192	Default RGB value, customizable by the system	
10	9	166	202	240	Default RGB value, customizable by the system	
11	246	255	251	240	Default RGB value, customizable by the system	
12	247	160	160	164	Default RGB value, customizable by the system	
13	248	128	128	128	Dark Gray	
14	249	255	0	0	Red	
15	250	0	255	0	Green	
16	251	255	255	0	Yellow	
17	252	0	0	255	Blue	
18	253	255	0	255	Magenta	
19	254	0	255	255	Cyan	
20	255	255	255	255	White	

4.3.1.2 Traveler Reserved Colors

To ensure that avatars have consistent appearances in all spaces and that their colors do not change when they move from one space to another, Traveler uses an almost fixed 192 colors for all spaces. 192 entries of the palette are used by 24 reserved colors that are ramped from light to dark with eight different intensities. The RGB value of the lightest entry of the ramp is fixed for all 24 colors. The level of lightness for the other seven entries is determined by the lighting intensity in the space. These 192 colors along with other colors in the palette can be used by 3D objects, texture maps, and sprites, as well as avatars, of course.

By using the colors specified in the ‘Original Colors’ columns of Table 5, you can make the final appearance of your space in Traveler more predictable. The second set of columns in the table shows the actual colors in the Traveler palette after the original colors specified in the VRML files are parsed and processed by Traveler.

If you are using 3D Studio MAX to create your space, you can use the supplied material library, *Dstrav.mat*, to assign pre-defined colors that match Traveler base colors to some or all of the 3D surfaces. Materials with single letter prefix (e.g., A.BEIGE DARK) can be used for flat and smooth shaded objects, and materials with double letter prefix (e.g., AA.BEIGE DARK) are used for self-illuminant objects.

The following table lists the RGB values of the Traveler 24 base colors:

No.	3D Studio Material	Original Colors Assigned to Objects			Original Colors Mapped to These Colors in Traveler		
		R	G	B	R	G	B
1	W/WW.WHITE WARM	243	230	214	246	236	224
2	B/BB.BEIGE LIGHT	215	166	120	225	187	148
3	I/II.BLUE LIGHT	134	140	227	161	166	234
4	4 X/XX.YELLOW	206	179	76	218	198	107
5	M/MM.GREEN LT	143	189	128	168	205	155
6	A/AA.BEIGE DARK	198	140	106	212	166	136
7	Q/QQ.ORANGE	214	135	60	224	161	90
8	N/NN.GREY COOL	118	118	149	147	147	173
9	C/CC.BEIGE LTBRWN	181	112	83	199	141	114
10	D/DD.BEIGEMEDBRN	170	99	67	190	129	98
11	O/OO.LIP-LT RED	152	44	52	176	72	81
12	H/HH.BLUE DARK	54	56	114	84	86	143
13	S/SS.RED DARK	143	34	34	168	60	60
14	U/UU.BROWNLIGHT	115	56	34	144	86	60
15	R/RR.PURPLE DARK	84	28	89	115	52	120
16	P/PP.LIP MED RED	105	3	60	135	56	90
17	V/VV.BRWNMED	94	44	28	125	72	52
18	L/LL.GREEN DK	39	73	39	66	104	66
19	J/JJ.BROWN GOLD	88	52	0	119	81	0
20	T/TT.SKIN-BROWNDK	74	34	25	105	60	48
21	K/KK.BROWN MED	44	28	11	72	52	27
22	G/GG.BLAK PRPLMED	21	11	21	42	27	42
23	F/FF.BLAK BLUE	11	11	28	27	27	52
24	E/EE.BLACK JET	25	0	11	48	0	27

4.3.1.3 Customizable Colors

The remaining 44 entries in the palette are completely customizable and vary from one space to another. Traveler extracts these 44 colors from the background images to fill the rest of the palette. So if you need to use a specific color that is not in Windows or Traveler segments, you can include it in the background images. This segment of palette can be used by all visual elements in the space.

4.3.2 How 3D and 2D Objects Use Available Colors

The following sections describe in more detail how the 3D or 2D objects use the available colors used in your space design.

3D Objects

When you assign any colors to 3D objects in your space, Traveler maps them to the closest match among all 256 colors in the palette.

Texture Maps

Like 3D objects, Traveler maps the colors in the texture map bitmaps to the closest match among all 256 colors in the palette. If there are more than 128 colors in the texture map bitmaps, Traveler reduces them to 128 first and then does the mapping.

Sprites

The number of colors used by a sprite is specified by the `colorCount` field in the corresponding `Dspace_Sprite` node. Traveler reduces the number of colors in the image, if needed, and then does the mapping like the texture maps.

Background Images

Traveler extracts the colors used in background images to fill 44 palette entries. If there are more than 44 colors in the background bitmaps, Traveler first reduces them to 44 colors. Unlike texture maps and sprites, the number of palette colors that background images can use does not exceed those 44 colors.

Important

You usually get a better result if you use image editing applications and

utilities such as Adobe Photoshop or DeBabelizer to reduce the number of colors in your bitmap images and/or to unify their color palettes instead of letting Traveler do an automatic color reduction for you.

4.4 Additional Considerations

This section discusses a few additional technical considerations.

4.4.1 Surface Normals and Back-face Culling

All polygons are single sided, meaning one side is shaded and the other side is completely transparent or invisible.

The visible surfaces should have consistent surface normal orientations for proper back-face culling. In some modeling tools, the user has little or no control over polygon normals or vertex ordering, so the user may have to flip the surface normals manually. It is therefore recommended that the designer test their designed space with Traveler to locate the erroneous polygons.

4.4.2 Background Images and Boundaries of Space

Bitmaps can be used with a special texture background element in the horizon that can map a set of images on an "infinitely far away sphere". These images are not drawn in perspective. No parallax occurs from this texture element so it must be designed to be used in limited ways.

Since background images are not in perspective, a space should be appropriately bounded, using `Dspace_CubeBoundary` node, so avatars cannot position themselves where all they see is the background. When only the background is in view, there is no perceived sense of 3D movement. This results in loss of orientation and should be avoided at all costs.

4.4.3 Avoiding Depth Buffer Artifacts

If two polygons face the same direction and are very close to each other, they may be rendered with undesirable visual effects. This is because their close proximity is smaller than the resolution of the depth buffer and the renderer is unsure which polygon should be drawn in front of which. This rendering artifact worsens the farther the camera is from the problem polygons. To avoid this problem, increase the space between the polygons.

4.5 Audio Authoring

Digital Space Traveler allows multiple users to enter rich graphical virtual worlds and communicate real-time using their voice instead of typing. While creating these virtual environments, authors can provide environmental audio scores which add value to the experience by playing two distinct roles:

- *Aesthetic* - Ambient audio and localized sounds greatly enhances the

depth of the experience for the user by reinforcing the theme of the environment.

- *Functional* - Ambient audio beds increase the perception of voice quality by applying a natural dithering effect on the voice waveform.

4.5.1 Definitions

RAS Random Audio Sequencer utilizing an eight channel randomizing audio engine.

Audio Channel One of eight total simultaneous sound generators used byRAS.

Audio Sequence A complete set of inner and outer loop randomization parameters.

Bed A looped sample with no delay between playback. For example, the background sound underneath flocking seagulls' sounds might be the sound of the rumbling ocean waves. It plays continuously until the space has been exited. To create a continuous looping bed; set -1 in the inner loop count range and a -1 in the outer loop count range.

Inner loop A set of the parameters defining the playback behavior of a sequenced audio event. Example: the "chirping" sound will playback and loop 1 to 5 times per event and have a 250 to 750 millisecond randomized delay between samples.

Outer loop The superset of the randomized looping function defining time between inner loop event playback. Example: The "chirping" inner loop audio event will playback once every 15 to 20 seconds and will continue infinitely until the space has been exited.

Score A complete ambient audio environment utilizing a soundtrack or randomized wav file playback approach.

4.5.2 Authoring Approaches

We have found that by establishing an ambient "bed" of looping audio content, audio is better able to meet aesthetic and functional goals. With an eye towards keeping audio memory requirements to a minimum, Digital Space developed the 8 channel RAS (Random Audio Sequencer) tool (i.e., *ras.exe*). RAS plays back audio using inner and outer nested loops with delay parameters yielding the advantage of non-repetitive audio content.

4.5.3 The Digital Space RAS Audio Authoring Tool

The RAS tool for Traveler is an interface to an 8-channel randomized audio playback engine used to create ambient audio soundscapes for Digital Space Traveler spaces. RAS allows the author to load *.wav or *.olw files into an audio channel parameter list and define either fixed or min/max time boundaries for randomized playback. RAS will save out the Digital Space VRML sound node with the custom RAS parameters and default 3D settings.

Traveler must be installed in order for RAS to run. RAS can be executed from any local directory.

4.5.3.1 Using RAS

To use the RAS tool, follow these steps:

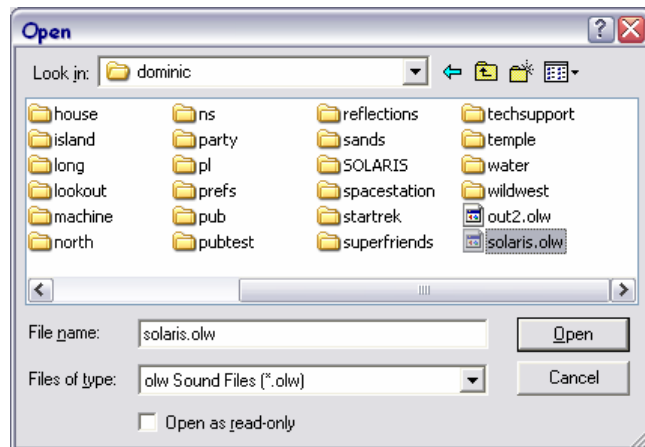
1. Double-click on the *ras.exe* executable file. If the file is on your desktop, the icon will look like this:



Ras.exe

The RAS tool is launched, and Digital Space Access Manager is loaded automatically.

2. From the RAS user interface, click on the “Select” button in the [Channel 1] panel, and select an audio file to play. RAS defaults to a compressed *.olw file masking when selecting a file for a voice channel.



If you have uncompressed *.wav files, you can select them by using the “File of type” option in the load dialog box. If you only have *.olw files,

you can convert them to decompress them to *.wav files using the “OLW to WAV” command on the Tools menu of the interface.

3. Click on the [Play] button in Channel 1 to play that channel. The [Play] button toggles to stop when a file is playing.

4. Adjust the audio parameters for the Channel 1 audio playing to desired positions. This is usually best done “on the fly” with a trial and error approach until the audio plays back as you would like.

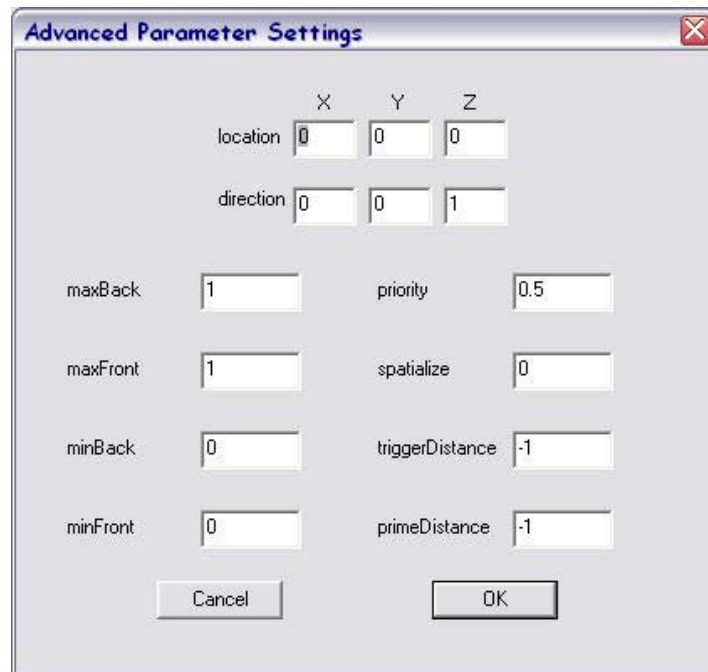
Adjust the volume settings for the entire score or individual channels, as appropriate, using the Master Volume and Channel Volume sliders, respectively.

Refer to “Definitions” on page 51 to understand the context of the parameters better.

5. Add additional *.wav (or *.olw) files to build the score as you would like. Following is an example of an audio score that has been built through adding multiple audio files, and adjusting their playback parameters as desired.



6. For advanced and very precise audio manipulation, click on the [Advanced] button for a specific *.wav (or *.olw) file that you have loaded into a channel. Following is an example of an “Advanced Parameter Settings” window that has been modified with precise values.



7. When you have completed the RAS score designing, from the File menu, select the “Save As” option to save the score as VRML code for implementation in Traveler.

4.5.3.2 RAS Audio Parameter Definition

Upon outputting your score to VRML, the parameters you set through inputting values in the RAS interface, or dragging the adjustable sliders are equated to VRML fields within VRML code. For a more comprehensive understanding of RAS’s audio parameters, refer to section A.2, “Digital Space VRML Extensions,” on page 64. Within the `Dspace_AmbientAudio` and `Dspace_Sound` node sections, the audio parameters are defined. Further, the “VRML Audio Example” on page 57 illustrates a VRML file outputted by RAS.

4.5.4 Audio Specifications for Traveler Digital Space Sound Node

Audio settings for a Traveler space is defined by the fields of the `Dspace_Sound` node and `Dspace_AmbientAudio` node. The parameters of these nodes are adjusted through the attenuation sliders in the RAS user interface, whereby the exact values of the audio adjustments are included in the appropriate nodes and fields in VRML output. Refer to Table 3, “Traveler 2.0 VRML Quick Reference,” on page 17 for specifics on these nodes.

File Format

All Traveler audio events use either `*.wav` or `*.olw` files in either 8 or 16-bit depths, mono or stereo. Sample rates must be 8 Khz or 16 Khz to mix with the voice codec.

Compression

Traveler can play back the proprietary `*.olw` compressed files yielding 15:1 compression on 16 Khz, or 16-bit `*.wav` files. Source files are compressed with the “WAV to OLW” function in the RAS tool. This compression technique is lossy, so for sensitive material like music, designers might consider comparing 8Khz 8-bit uncompressed files to compressed files and choose the best results against the memory tradeoff.

4.5.5 VRML Audio Example

Following is an example of the audio part of VRML code for a Traveler space.

```
Dspace_Sound {  
  fields [  
    SFString url,  
    SFVec3f location,  
    SFVec3f direction,  
    SFFloat maxBack,  
    SFFloat maxFront,  
    SFFloat minBack,  
    SFFloat minFront,  
    SFFloat intensity,  
    SFFloat priority,  
    SFLong spatialize,  
    SFLong innerLoopsMin,  
    SFLong innerLoopsRange,  
    SFFloat innerLoopsDelayMin,  
    SFFloat innerLoopsDelayRange,  
    SFLong outerLoopsMin,  
    SFLong outerLoopsRange,  
    SFFloat outerLoopsDelayMin,
```

```
        SFFloat outerLoopsDelayRange,  
        SFFloat triggerDistance,  
        SFFloat primeDistance ]  
url "" # SFString (range: any)  
location 0 0 0 # SFVec3f (range: any)  
direction 0 0 1 # SFVec3f (range: any unit vector)  
maxBack 1 # SFFloat (range: >= 0)  
maxFront 1 # SFFloat (range: >= 0)  
minBack 0 # SFFloat (range: < maxBack)  
minFront 0 # SFFloat (range: < maxFront)  
intensity 1 # SFFloat (range: 0-1)  
priority 0.5 # SFFloat (range: 0-1)  
spatialize 0 # SFLong (range: 0, 1 or 2)  
innerLoopsMin 1 # SFLong (range: >= 0)  
innerLoopsRange 0 # SFLong (range: >= 0)  
innerLoopsDelayMin 0 # SFFloat (range: >= 0)  
innerLoopsDelayRange 0 # SFFloat (range: >= 0)  
outerLoopsMin 1 # SFLong (range: >= 0)  
outerLoopsRange 0 # SFLong (range: >= 0)  
outerLoopsDelayMin 0 # SFFloat (range: >= 0)  
outerLoopsDelayRange 0 # SFFloat (range: >= 0)  
triggerDistance -1 # SFFloat (range: -1 = maxFront, >  
0 is the  
trigger distance)  
primeDistance -1 # SFFloat (range: -1 = maxFront, > 0  
is the  
prime distance)
```

5 • • • Additional Resources

The following sections include pointers for a 3D designer to further assist in learning more about 3D, VRML, and associated technologies. All specified URLs were accurate at the time of this printing, but are subject to change. If you are not able to locate the references through the given URLs, conduct an "Internet Search".

5.1 Introduction Material

A Beginner's Guide to VRML, Netscape Communications Corporation, 1996.

http://www.netscape.com/eng/live3d/howto/vrml_primer_index.html

This guide provides a ground-level introduction to VRML for those wanting to author VRML worlds for use on the Internet. It outlines basic concepts necessary to begin designing and authoring VRML worlds. It assumes that the reader has no background in 3D graphics, but has some familiarity with the Internet. This document also lists other resources which may be helpful to VRML authors.

3D Studio Max to VRML Tutorial, dFORM Inc.

<http://www.dform.com/inquiry/tutorials/3dsmax/>

This tutorial is designed to show how to create and export VRML files through 3D Studio MAX and its VRML exporter. The exporter translates many of the features of 3D Studio MAX into VRML 1.0, 2.0 and VRBL (Virtual Reality Behavior Language) files. VRBL is an extension to VRML 1.0.

5.2 More Information on VRML

The VRML Consortium

<http://www.vrml.org/>

The VRML Consortium, Inc. is a nonprofit corporation which evangelizes VRML as the open standard for 3D multimedia and shared virtual worlds on the Internet. The VRML Consortium Web site offers expansive information about VRML, its developer community, opportunities within the VRML community, and information about VRML interoperability, among others.

VRML Repository

<http://dssc.edu/vrml/>

The VRML repository is a comprehensive resource for disseminating information relating to VRML. The VRML repository is maintained by the San Diego Supercomputer Center (DSSC).

SGI VRML Site

<http://vrml.sgi.com/intro.html>

The SGI VRML site includes information about VRML as it pertains to Silicon Graphics, Incorporated, and otherwise.

5.3 Standards

Open Inventor (by Silicon Graphics, Inc.)

<http://www.sgi.com/Technology/Inventor.html>

This Web site includes an overview of the relationship between Open Inventor and VRML. Open Inventor™ is an object-oriented toolkit for developing interactive, 3D graphics applications. It also defines a standard file format for exchanging 3D data among applications. Open Inventor serves as the basis for the VRML specification.

Uniform Resource Locators (URL) Specifications

<http://ds.internic.net/rfc/rfc1738.txt>

This document specifies a Uniform Resource Locator (URL), the syntax and semantics for a compact string representation for a resource available via the Internet.

Relative Uniform Resource Locators Specifications

<http://ds.internic.net/rfc/rfc1808.txt>

This document describes the syntax and semantics for “relative” Uniform Resource Locators (relative URLs): a compact representation of the location of a resource relative to an absolute base URL. It is a companion to RFC 1738, “Uniform Resource Locators (URL),” which specifies the syntax and semantics of absolute URLs.

PNG (Portable Network Graphics) Specifications

<http://www.w3.org/pub/WWW/TR/REC-png.html>

This document describes PNG (Portable Network Graphics), an extensible file format for the lossless, portable, well-compressed storage of raster images. PNG is the most commonly used bitmap file format in Digital Space Traveler spaces.

5.4 More Readings

Reading List from the MIT graduate seminar, Digital Communities: Urban Planning and Design in Cyberspace

<http://alberti.mit.edu/arch/4.207/readings.html> (also see main Web site page, at <http://alberti.mit.edu/arch/4.207/homepage.html>).

A ••• VRML Specifications

A.1 VRML Standards

Following are the specific references for VRML specifications that this document, and ultimately Digital Space Traveler space development, is based on:

The Virtual Reality Modeling Language (VRML), Version 1.0C Specification

<http://vag.vrml.org/vrml10c.html>

Traveler 2.0 is compliant with VRML 1.0 specification. Refer to section 2.7 of this document, Traveler 2.0 VRML Quick Reference, for details on Traveler's VRML 1.0 implementation.

The Virtual Reality Modeling Language (VRML), Version 1.1 Draft Specification

<http://vag.vrml.org/vrml-1.1.html>

This version of VRML was never finalized, and the development efforts to complete this document were replaced by much broader and more extensive VRML 2.0 efforts. Currently the official VRML releases are 1.0 and 2.0

Traveler supports only three VRML 1.1-specific nodes to enhance Digital Space spaces with additional features. They are the `Background`, `Environment`, and `WorldInfo` nodes. These nodes should be treated as Digital Space extensions to VRML, with written out descriptions of their fields each time they are used.

Refer to Table 3, "Traveler 2.0 VRML Quick Reference," on page 17 for details on Traveler's VRML 1.1 Draft implementation, and the *Extensibility* section of the VRML 1.0C specification for more information on nonstandard nodes.

A.2 Digital Space VRML Extensions

Traveler 2.0 is compliant with VRML 1.0 specification. Its VRML implementation corresponds with the specification, except as noted in Table 2.7, "Traveler 2.0 VRML Quick Reference". Traveler also supports three VRML 1.1

Draft nodes, as specified in the same table.

Traveler uses extensions to VRML to support multi-participant, real-time voice communications, community building, and socialization in 3D spaces.

There are a total of 11 Digital Space nodes. According to VRML specification, it is legal to add new nodes for specific VRML implementation, such as

Digital Space Traveler, as long as their fields specification are written out with every instance of these nodes, whether or not those node types were previously used in the scene graph. This lets other VRML browsers parse and ignore the extensions without causing errors.

The fields specification are written after the opening curly-brace for the node, and consist of the keyword 'fields' followed by a list of the types and names of fields used by that node, all enclosed in square brackets and separated by commas. For example, `Dspace_CubeBoundary` node is written like this:

```
Dspace_CubeBoundary {  
    fields [ SFVec3f min,  
            SFVec3f max ]  
    min 0 0 0  
    max 1 1 1  
}
```

Refer to the *Extensibility* section of the VRML 1.0C specification for more information on non-standard nodes.

Traveler uses both `.wrl` and `.dsv` file extensions for VRML files, with `.dsv` as the default one, to avoid any conflicts with other VRML browsers that users might run on their computers. In terms of the content and functionality, there is no difference between an `.dsv` file and `.wrl` file.

Dspace_AmbientAudio

This node defines an ambient sound source that can be heard anywhere in the space at a constant level. `Dspace_Sound` node has all functionalities of this node plus several additional features. There can be up to eight total ambient sound generators (`Dspace_AmbientAudio` and `Dspace_Sound` set to ambient) specified in a space.

The `url` field specifies the URL of a sound file to play.

The `innerLoopMinCount` field specifies the minimum times a sample is played back during an audio event. A negative number creates an infinite forward loop.

The `innerLoopCountRange` field specifies the number in addition to the

`innerLoopMinCount` defining the maximum times a sample will playback during an audio event.

The `innerLoopMinDelay` field specifies the minimum delay amount in milliseconds between sample playback in an audio event.

The `innerLoopDelayRange` field specifies the amount of delay in milliseconds in addition to `innerLoopMinDelay` defining the maximum delay time between sample playback in an audio event.

The `outerLoopMinCount` field specifies the minimum times an inner loop is played back during an audio sequence. A negative number creates an infinite forward loop.

The `outerLoopCountRange` field specifies the number in addition to the `outerLoopMinCount` defining the maximum times a inner loop will playback during an audio sequence.

The `outerLoopMinDelay` field specifies the minimum delay amount in milliseconds between inner loop playback in an audio sequence.

The `outerLoopDelayRange` field specifies the amount of delay in milliseconds in addition to `outerLoopMinDelay` defining the maximum delay time between inner loop playback in an audio sequence.

The `leftVolume` and `rightVolume` fields specify the volumes of left and right output.

Syntax

```
DSpace_AmbientAudio {
    fields [ SFString url,
             SFLong innerLoopMinCount,
             SFLong innerLoopCountRange,
             SFFloat innerLoopMinDelay,
             SFFloat innerLoopDelayRange,
             SFLong outerLoopMinCount,
             SFLong outerLoopCountRange,
             SFFloat outerLoopMinDelay,
             SFFloat outerLoopDelayRange,
             SFFloat leftVolume,
             SFFloat rightVolume ]

    url ""
    innerLoopMinCount -1
    innerLoopCountRange 0
    innerLoopMinDelay 0
    innerLoopDelayRange 0
    outerLoopMinCount -1
    outerLoopCountRange 0
    outerLoopMinDelay 0
```

```
outerLoopDelayRange 0
leftVolume 1.0
rightVolume 1.0
}
```

Dspace_CubeBoundary

This node defines a box that bounds avatar/camera movement in space. There can be no more than one `Dspace_CubeBoundary` node in the entire scene graph. By default, there is no bounding box in the space and avatar/camera movement is unrestricted.

The `min` and `max` fields specify the diagonally opposite corners of the bounding box.

Syntax

```
Dspace_CubeBoundary {
    fields [    SFVec3f min,
               SFVec3f max ]
               min 0 0 0
               max 1 1 1
}
```

Dspace_EntryPoint

This node defines attributes of avatar's entry to a space such as landing position and initial orientation. It also defines an optional flying sequence between two specified points along a spiral path as the initial approach to a space. The position of entry point is relative to the current state of geometric transformation. Therefore, for absolute landing zones, place the node at the beginning of the source file.

There can be multiple entry points defined; they are selected at random by incoming avatars. The `PerspectiveCamera` node is considered as a limited version of `Dspace_EntryPoint` node, so both can be used as entry points definitions. The actual landing spot is selected at random within the cube defined by `width`, `height`, and `depth` fields.

By default, if there are no `Dspace_EntryPoint` or `PerspectiveCamera` nodes defined in a space, the avatars land at the origin (0,0,0) facing toward negative Z direction.

The `position` field specifies the X, Y, and Z coordinates of the landing point.

The `rotation` field specifies the initial orientation or direction of the avatar.

The `width`, `height`, and `depth` fields define a cube within which the landing spot is selected at random for each entry point. Setting all these three fields to 0 turns off the randomization.

The `fly` field toggles the flying sequence. If this field is set to `FALSE`, the settings of the subsequent fields, `startPosition` and `startRotation` will be ignored.

The `startPosition` field specifies the X, Y, and Z coordinates of the starting point of the flying sequence. This field is ignored if the `fly` field is set to `FALSE`.

The `startRotation` field specifies the starting orientation or direction of the avatar at the beginning of the flying sequence. This field is ignored if the `fly` field is set to `FALSE`.

Syntax

```
Dspace_EntryPoint {
    fields [ SFVec3f position,
            SFRotation rotation,
            SFFloat width,
            SFFloat height,
            SFFloat depth,
            SFBool fly,
            SFVec3f startPosition,
            SFRotation startRotation ]

    position 0 0 0
    rotation 0 0 -1 0
    width 1
    height 0
    depth 1
    fly TRUE
    startPosition 0 10 0
    startRotation 1 0 0 -1.5
}
```

Dspace_Horizon

This node specifies a background texture at infinite distance. It forms a panorama that is placed behind all objects in the space and in front of the ground and sky. There can be multiple `Dspace_Horizon` nodes defined in order to tile several background images together.

The `url` field specifies the URL of the bitmap image that defines the background panorama.

The `angularWidth` field specifies the angle of horizon that the image occupies. For example, a value of 3.142 radians for this field, specifies that the image takes up half the horizon (e.g. from due East to due West).

The `bearing` field specifies an angle at which the lower left edge of the image is placed at the horizon.

The `elevation` field indicates in radians how "high" above the horizon the image should be. A value of 0 means the bottom of the element is resting on the virtual horizon.

Syntax

```
Dspace_Horizon {
    fields [ SFString url,
             SFFloat angularWidth,
             SFFloat bearing,
             SFFloat elevation ]
    url ""
    angularWidth 3.142
    bearing 0
    elevation 0
}
```

Dspace_MaxAvatars

This node specifies maximum number of avatars allowed in an instance of a space. There should be only one `Dspace_MaxAvatars` node in the entire scene graph. If the value of the `max` field is set to 0, then the determination of maximum number of avatars is made by the default setting of the Digital Space community server.

Syntax

```
Dspace_MaxAvatars {
    fields [ SFLong max ]
    max 0
}
```

Dspace_MaxInstances

This node specifies maximum number of instances or clones of a space that can be created in a community. There should be only one `Dspace_MaxInstances` node in the entire scene graph. To turn off space cloning, set the `max` field to 1. If the value of the `max` field is set to 0, then the determination of maximum number of clones is made by the default setting of the Digital Space community server.

Syntax

```
Dspace_MaxInstances {  
    fields [ SFLong max ]  
    max 0  
}
```

Dspace_Server

This required node specifies the name of Digital Space access server which the space is connected to. There should be only one `Dspace_Server` node in the entire scene graph.

The `server` field specifies the name of Digital Space access server. If this field is set to `NO_CONNECT`, the single-user mode of Traveler is enabled.

The `url` field specifies the URL of the main `.dsv/wrl` file for the space (i.e. the file that contains this node).

Syntax

```
Dspace_Server {  
    fields [ SFString server,  
            SFString url ]  
    server ""  
    url ""  
}
```

Dspace_Sound

This node defines a sound source and its positioning and spatial presentation in a space. The `Dspace_Sound` node implements the functionality of the VRML 2.0 sound node with several added features.

The sound may be located at a point and emit sound in a spherical or ellipsoid pattern, in the local coordinate system. The ellipsoid is pointed in a particular direction and may be shaped to provide more or less directional focus from the location of the sound. The `Dspace_Sound` node may also be used to describe an ambient sound which tapers off at a specified distance from the sound node.

The `Dspace_Sound` node also enables ambient background sound to be created by setting of the `maxFront` and `maxBack` to the radius of the area for the ambient noise. If ambient sound is required for the whole scene then these values should be set to at least cover the distance from the location to the

farthest point in scene from that point (including effects of transforms).

Digital Space has added two *ellipses*, specified by `triggerDistance` and `PrimeDistance` fields, to enhance the playback behavior of the sound node.

The `Dspace_Sound` node can also be set to spatialize local audio. The result is stereo panning and distance attenuation of a sound file in relationship to the coordinates of an avatar.

The `url` field specifies the URL of a sound file to play.

The `location` field specifies the position (center) of the sound source.

The `minBack` and `minFront` fields determine the extent of the full intensity region behind and in front of the sound (around the location of the sound emitter). If the location of the sound is taken as a focus of an ellipsoid, the `minBack` and `minFront` values, in combination with the `direction` vector determine the two foci of an ellipsoid bounding the ambient region of the sound.

Similarly, `maxBack` and `maxFront` fields determine the limits of audibility behind and in front of the sound; they describe a second, outer ellipsoid. If `minFront` equals `minBack` and `maxFront` equals `maxBack`, the sound is omni-directional, the `direction` vector is ignored, and the min and max ellipsoids become spheres centered around the sound node. The fields `minFront`, `maxFront`, `minBack`, and `maxBack` are scaled by the parent transformations. Their values must be equal or larger than 0.

The inner ellipsoid defines a space of full intensity for the sound. Within that space the sound will play at the intensity specified in the sound node. The outer ellipsoid determines the maximum extent of the sound. Outside that space, the sound cannot be heard at all. In between the two ellipsoids, the intensity drops off proportionally with inverse square of the distance. With this model, a sound usually will have smooth changes in intensity over the entire extent in which it can be heard. However, if at any point the maximum is the same as or inside the minimum, the sound is cut off immediately at the edge of the minimum ellipsoid.

The `intensity` field specifies the volume of the sound source. It's a floating point number that ranges from 0.0 to 1.0. An intensity of 0 is silence, and an intensity of 1 is the full volume of the sound source.

The `priority` field gives the author some control over which sounds Traveler will choose to play when there are more sounds active than sound channels available. The priority varies between 0.0 and 1.0, with 1.0 being the highest priority.

The `spatialize` field is used to spatialize local audio. The result is stereo panning and distance attenuation of a sound file in relationship to the coordinates of an avatar. Sounds can also be set to ambient, which is not localized and plays at a constant level throughout the VRML scene. The field value of 0 means no spatialization. The sound, in this case, is attenuated by the Traveler's Ambient slider. The field value of 1 means that the spatialization is performed and the sound is attenuated by the Traveler's Effects slider. The field value of 2 also means that the spatialization is performed, but the sound is attenuated by the Traveler's Ambient slider.

The `innerLoopsMin` field specifies the minimum times a sample is played back during an audio event. A negative number creates an infinite forward loop.

The `innerLoopsRange` field specifies the number in addition to the `innerLoopMinCount` defining the maximum times a sample will playback during an audio event.

The `innerLoopsDelayMin` field specifies the minimum delay amount in milliseconds between sample playback in an audio event.

The `innerLoopsDelayRange` field specifies the amount of delay in milliseconds in addition to `innerLoopMinDelay` defining the maximum delay time between sample playback in an audio event.

The `outerLoopsMin` field specifies the minimum times an inner loop is played back during an audio sequence. A negative number creates an infinite forward loop.

The `outerLoopsRange` field specifies the number in addition to the `outerLoopMinCount` defining the maximum times a inner loop will playback during an audio sequence.

The `outerLoopsDelayMin` field specifies the minimum delay amount in milliseconds between inner loop playback in an audio sequence.

The `outerLoopsDelayRange` field specifies the amount of delay in milliseconds in addition to `outerLoopMinDelay` defining the maximum delay time between inner loop playback in an audio sequence.

The `triggerDistance` field defines an ellipse, allowing the author to start a sound at a different location than the outer ellipse. An example might be an alarm that is triggered inside the `maxFront` (`Back`) and is audible to the user until the `maxFront` (`Back`) ellipse has been exited.

The `primeDistance` field defines an ellipse, when crossed by an avatar, resets the sound node to play again when the `triggerDistance` ellipse is

crossed. By setting this ellipse to outside the bounding box of the VRML scene, the author can ensure a single playback instance of that sample. This may be useful with voice samples where the author does not want the content repeated.

Syntax

```
Dspace_Sound {
  fields [
    SFString url,
    SFVec3f location,
    SFVec3f direction,
    SFFloat maxBack,
    SFFloat maxFront,
    SFFloat minBack,
    SFFloat minFront,
    SFFloat intensity,
    SFFloat priority,
    SFLong spatialize,
    SFLong innerLoopsMin,
    SFLong innerLoopsRange,
    SFFloat innerLoopsDelayMin,
    SFFloat innerLoopsDelayRange,
    SFLong outerLoopsMin,
    SFLong outerLoopsRange,
    SFFloat outerLoopsDelayMin,
    SFFloat outerLoopsDelayRange,
    SFFloat triggerDistance,
    SFFloat primeDistance ]

  url "" # SFString
  location 0 0 0 # SFVec3f
  direction 0 0 1 # SFVec3f
  maxBack 1 # SFFloat
  maxFront 1 # SFFloat
  minBack 0 # SFFloat
  minFront 0 # SFFloat
  intensity 1 # SFFloat
  priority 0.5 # SFFloat
  spatialize 0 # SFLong
  innerLoopsMin 1 # SFLong
  innerLoopsRange 0 # SFLong
  innerLoopsDelayMin 0 # SFFloat
  innerLoopsDelayRange 0 # SFFloat
  outerLoopsMin 1 # SFLong
  outerLoopsRange 0 # SFLong
  outerLoopsDelayMin 0 # SFFloat
  outerLoopsDelayRange 0 # SFFloat
  triggerDistance -1 # SFFloat
  primeDistance -1 # SFFloat
}
```

Dspace_Spin

This node specifies a constant 3D spinning about an arbitrary axis through the origin. The spinning affects all subsequent nodes within the scope of the `Dspace_Spin` node.

Syntax

```
Dspace_Spin {  
    fields [ SFRotation rotation ]  
    rotation 0 1 0 0  
}
```

Dspace_Sprite

This node represents a 2D bitmap that always faces the viewer. It is always aligned with the camera, and is therefore best suited to displaying spherically or cylindrically symmetric objects.

The `url` field specifies the URL of a bitmap file to be used as the image source.

The `colorCount` field specifies the maximum number of colors from 256 palette entries the sprite will use when in view. If the actual number of colors in the source image is more than the specified number, Traveler does an automatic color reduction for the bitmap.

The `width` field specifies the width of the sprite. If the value of this field is set to 0 and a non-zero height is specified, the width of the sprite is automatically calculated by using the original aspect ratio of the image.

The `height` field specifies the height of the sprite. If the value of this field is set to 0 and a non-zero width is specified, the height of the sprite is automatically calculated by using the original aspect ratio of the image.

If both `width` and `height` fields are set to non-zero values and their aspect ratio is different than the original one, the sprite is automatically stretched to new dimensions. Specifying a width and height of 0 indicates that the object does not scale with distance at all.

Syntax

```
Dspace_Sprite {  
    fields [ SFString url,  
            SFLong colorCount,  
            SFFloat width,  
            SFFloat height ]  
    url ""  
    colorCount 128
```

```
width 1
height 1
}
```

Dspace_UserPortal

This node sets the subsequent `WWWAnchor` to be a user portal, instead of a regular portal. For example:

```
Separator {
  Dspace_UserPortal {
    fields []
  }
  WWWAnchor {
    name "user.dsv"
    description "User Space"
    WWWInline { name "props/obj.wrl" }
  }
}
```

Syntax

```
Dspace_UserPortal {
  fields []
}
```

B • • • Converting DSS Spaces to VRML

B.1 Necessary Tools

To convert DSS spaces to VRML, you will need the following software installed:

- *Kinetix 3D Studio MAX®*, R1.2 or later, as the primary conversion tool
- *3D Studio MAX VRML Export* plug-in, v1.0 or later
- *Kinetix 3D Studio R4* (if .3ds files are saved last in this version)
- An image editor that supports PNG, BMP, and JPEG file formats (e.g., *JASC Paint Shop Pro, v4.0*)
- A text editor
- Digital Space Traveler 2.0 to view and test spaces

For more information on these software packages, refer to their Web pages describing or distributing this software.

B.2 Conversion Process

The general procedure for converting DSS spaces to VRML includes the following steps:

- Organizing project files and directory trees to use relative URLs.
- Converting bitmap files to PNG format, a format with a better compression rate, and finer control over transparency information, than available with BMP or JPEG file formats.
- Converting .3ds files to VRML.
- Converting .dss file to VRML.

Note

The procedure described in the following sections outlines a simple and straight-forward way of converting DSS spaces to VRML. The resulting VRML file has similar data organization of the corresponding DSS file. This procedure is not to be considered the most efficient way of structuring a VRML scene graph.

It is very important to organize the many component files that make up your Digital Space Traveler space in a way that is easily referenceable (i.e., common directory structures) within the different files. For example, the entrance space files for the Digital Space Utopia community are organized on the server as follows:

```
http://anyserver.com/places2a/entrance.dsv
http://anyserver.com/places2a/maps/*.png
http://anyserver.com/places2a/props/*.wrl
http://anyserver.com/places2a/sprites/*.png
http://anyserver.com/places2a/texmaps/*.png
http://anyserver.com/places2a/wavs/*.olw
```

Create a similar file tree on your working computer. By doing so, the file tree from above example will look like the following:

```
c:\any_path\entrance.dsv
c:\any_path\maps/*.png
c:\any_path\props/*.wrl
c:\any_path\sprites/*.png
c:\any_path\texmaps/*.png
c:\any_path\wavs/*.olw
```

In this way, you can use relative URLs in your VRML files, unless the referenced data is located on a different server. For example, if entrance.dsv file is referencing a file in props directory, the relative URL would be:

```
WWWInline { name "props/aztec.wrl" }
```

B.2.1 Converting Bitmap Files to PNG

There are three main groups of bitmaps for Digital Space spaces: *background images*, *sprites*, and *texture maps*. The process to convert each group of bitmaps to PNG format is slightly different in each case. In all cases, however, you will need an image editor, such as *JASC Paint Shop Pro, v4.0* to convert the bitmaps to PNG format.

Converting Background Images to PNG

Since background images do not contain transparency information, follow these steps:

1. Open the .bmp file(s) within an image editor.
2. Save the file as an 8-bit non-interlaced PNG file with no transparency.

Background image PNG files should be saved to the `c:\any_path\maps\` directory.

Converting Sprites to PNG

If the sprite *does not contain* transparency information, follow these steps:

1. Open the .bmp file(s) within an image editor.
2. Save the file as an 8-bit non-interlaced PNG file with no transparency.

If the sprite *does contain* transparency information, with black being the transparent color, follow these steps:

1. Open the .bmp file(s) within an image editor.
2. Set the transparency value to black (RGB: 0, 0, 0).
3. Save the file as an 8-bit non-interlaced PNG file with transparency.

Sprite PNG files should be saved into the `c:\any_path\sprites\` directory.

Converting Texture Maps to PNG

If the texture map *does not contain* transparency information, follow these steps:

1. Open the .bmp file in an image editor.
2. Save the file as an 8-bit non-interlaced PNG file with no transparency.

If the texture map does contain transparency information, with opacity map saved as a separate .bmp file, follow these steps:

1. Open both the texture map and opacity map .bmp files within an image editor.

2. Change the color depth of the texture map file to 24-bit.
3. Add an alpha channel or mask to the texture map file, using the opacity map file as the source.
4. Save the composite image as a 32-bit non-interlaced PNG file with transparency.

Texture map PNG files should be saved into the `c:\any_path\texmaps\` directory.

B.2.2 Converting .3ds Files to VRML

The actual process of converting .3ds files to VRML consists of three parts:

- Re-saving .3ds files in 3D Studio R4 (if necessary).
- Converting .3ds files into .max files.
- Exporting .max files to VRML (.wrl files).

Re-saving .3ds Files in 3D Studio R4

As noted in the *3D Studio MAX* documentation, files that have been saved using the *3D Studio R4* “Save Selected” command do not import correctly into *3D Studio MAX*.

To make these .3ds files usable by *3D Studio MAX*, follow these steps:

1. Open the .3ds file from within *3D Studio R4*.
2. Save the file using the “Save” command.

Converting .3ds Files into .max Files

Follow these steps to convert the .3ds files into .max files, and then prepare the files to be converted into VRML files:

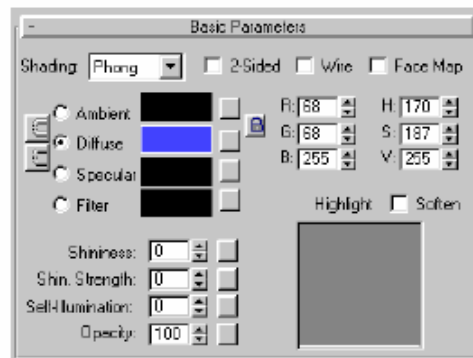
1. Open the .3ds file within *3D Studio MAX*, then save it as a .max file.
2. Delete all lights, cameras, and shapes.
3. Update the flat and smooth shaded materials in the scene by:
 - leaving the diffuse color as is
 - setting the ambient, specular, and filter colors to black (RGB:

0,0,0)

- setting the shininess, shininess strength, and self-illumination to 0
- unchecking 2-sided, wire, face map, and soften options
- setting the shading to Phong

Figure 1 shows the *3D Studio MAX* “Basic Parameters” dialog with the settings configured as described above.

Figure 1. Basic Parameters of Flat and Smooth Shaded Materials



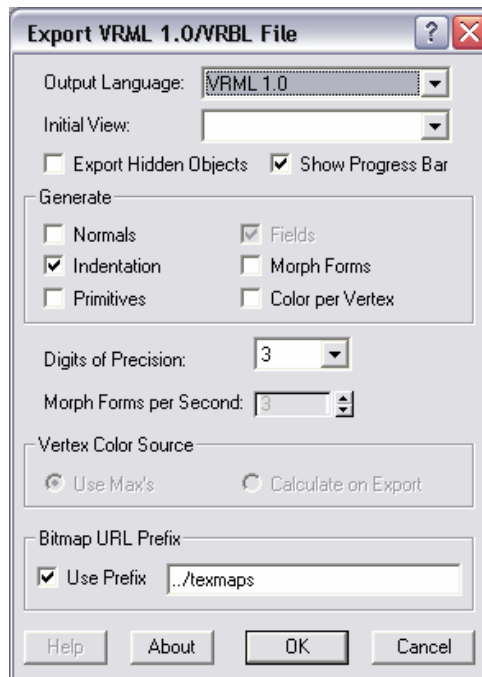
4. Update the self-illuminant materials in the scene, using the same settings as described in step 3 (for flat or smooth shaded materials) *except* that the self-illumination must be set to 100 (instead of 0).
5. Update the materials with texture maps, using the same settings as described in step 4 (for self-illuminant materials) *except* that the diffuse color needs to be set to white (RGB: 255, 255, 255). Also, the .bmp file needs to be replaced by the corresponding .png file (make sure that file name and extension are all in lower case).

Exporting .max Files to VRML

Follow these steps to export .max files to VRML:

1. Select the “Export...” command from the *File* menu within *3D Studio MAX*.
2. Select VRML (.wrl) as the type, enter a filename, then press the “Save” button. A dialog box similar to Figure 2 is displayed.

Figure 2. VRML Export Settings



3. Modify the settings in the dialog box, according to the following guidelines, then press the “OK” button.

- *Output Language* - Digital Space Traveler 2.0 is a VRML 1.0-compliant application. Select “VRML 1.0” from the list.
- *Initial View* - Since you have already deleted all the cameras, there should be nothing in this list. Leave this option blank.
- *Export Hidden Objects* - By default, hidden objects will not be exported. Check the “Export Hidden Objects” option only if you want to export hidden objects. Otherwise, leave unchecked.
- *Generate Normals* - Digital Space Traveler 2.0 automatically generates surface normals. Therefore, leave this option unchecked. Refer to section 4.1.2 for details on surface normals.
- *Generate Indentation* - Check this option to make the VRML source code easier to read and edit. Refer to section 4.1.2 for details on redundant data in VRML files.
- *Generate Primitives* - Leave this option unchecked. All imported objects from .3ds files are converted to meshes, so this option practically does not have any effect on them. If you have added new objects in your .max file, see section 4.1.1 for details on polygon count *before you set this option*.

- *VRML 2.0 Options* - This section is automatically disabled when you select VRML 1.0 as the output language.
- *Digits of Precision* - Select “3” to set the number of decimal points used for calculating dimensions. Refer to section 4.1.2 for details on redundant data in VRML files.
- *Bitmap URL prefix* - If your texture map files are in the same directory as your .wrl file, uncheck the “Use prefix” option.

If your files are organized in different directories (as suggested), check the “Use prefix” option and specify a relative path.

B.2.3 Converting DSS File to VRML

Finally, this section guides you through structuring VRML files derived from the main components of your .dss file. Refer to Table 1 on page 97 for a summary mapping of DSS to VRML code correlations. The following key components are described in further detail throughout this section:

- header and home address information
- paths
- global change of scale
- environmental properties of space
- light sources
- 3D and 2D objects, their properties, transformations, and animation
- ambient audio
- adding `DSpace_EntryPoint` node (new feature)

Header and Home Address Information

Creating a new file with your text editor. Save it with an .dsv file extension. This file will become the main VRML file for your space.

Insert a VRML header in the .dsv file derived from the header in your .dss file, using the syntax structure shown in the following code examples.

DSS The .dss file should have a header and a `HOME_ADDRESS` line similar to the following:

```
<DSS, 1.0 , "anyserver.com", "Carousel", 12, 0>
HOME_ADDRESS: "http://anyserver.com/places/
carousel.dss"
; rest of the file
```

The header specifies the script type (DSS), the DSS specification's version (1.0), the name of the Digital Space access server, the maximum number of avatars in the space, and the maximum number of clones of the space. The HOME_ADDRESS keyword specifies the URL or location of the main .dss file.

VRML Modify the .dsv file that you created, using a standard VRML 1.0 header format, plus four nodes to specify the above information, as shown:

```
#VRML V1.0 ascii
Separator {
  DSpace_Server {
    fields [ SFString server,
             SFString url ]
    server " anyserver.com"
    url "http://anyserver.com/places/
carousel.dsv"
  }
  WorldInfo {
    fields [ SFString title,
             MFString info ]
    title "VRML Carousel"
    info ""
  }
  DSpace_MaxAvatars {
    fields [ SFLong max ]
    max 12
  }
  DSpace_MaxInstances {
    fields [ SFLong max ]
    max 0
  }
  # rest of the file
}
```

Paths

DSS In the .dss file, there are two keywords for specifying the paths, BMP_PATH, specifying the location of the texture maps, and HOME_ADDRESS which was discussed in the previous section. The BMP_PATH line looks like the following:
BMP_PATH: "http://anyserver.com/places/texmaps"

VRML In VRML, you do not have to explicitly specify the location of the

texture maps. The `filename` field in `Texture2` node includes both the path and file name. *3D Studio MAX VRML Export* plug-in can automatically generate this information when you convert .3ds files to VRML. So if any of your .wrl files in the props directory contain texture maps, nodes similar to the following are automatically created:

```
Texture2 {
    filename "../texmaps/image.png"
}
```

Global Change of Scale

VRML In VRML spaces, one unit equals one meter, where in DSS spaces, one unit equals one inch. Therefore, when you convert an DSS space to VRML, the space will be approximately 40 times larger. To revert your space back to proper scale, insert a `Transform` node (in the .dsv file) with a scale factor of 0.025 before the sections that describe 3D and 2D objects. For example:

```
#VRML V1.0 ascii
Separator {
    # required nodes
    Transform { scaleFactor 0.025 0.025 0.025 }
    # rest of the file
}
```

Environmental Properties of Space

In this category, the following DSS keywords are either obsolete or replaced by built-in functions of Digital Space Traveler 2.0:

- `BACKGROUND_COLORS` - Digital Space Traveler sets this parameter to 44.
- `COLLISION` - Digital Space Traveler automatically turns collision detection *on* for portals and links (and no other objects).
- `FIELD_OF_VIEW` - Digital Space Traveler 2.0 fixes the field of view to 90 degrees in all VRML spaces.
- `TYPE` - This information is implied by the settings of the `Background` node.

Other space properties are converted to VRML as follows:

Bounding Box

DSS `BOUNDING_BOX: -1000.0, 0.0, -1000.0, 1000.0, 1000.0, 1000.0`

```
VRML DSpace_CubeBoundary {
    fields [ SFVec3f min,
             SFVec3f max ]
    min -1000.0 0.0 -1000.0
    max 1000.0 1000.0 1000.0
}
```

Ambient Lighting

```
DSS AMBIENT_LIGHT: 0.3, 0.3, 0.3
```

```
VRML Environment {
    fields [ SFFloat ambientIntensity ]
    ambientIntensity 0.3
}
```

Sky and Ground Colors

```
DSS TYPE: HALF
GROUND_COLOR: 0.5, 0.5, 0.5
SKY_COLOR: 0, 0, 1
```

```
VRML Background {
    fields [ MFColor groundColors,
             MFColor skyColors,
             MFString scenery ]
    groundColors [ 0.5 0.5 0.5 ]
    skyColors [ 0 0 1 ]
    scenery ""
}
```

Background Sky Texture

```
DSS SKY_TILE: "http://anyserver.com/places/maps/shspace.bmp"
```

```
VRML Background {
    fields [ MFColor groundColors,
             MFColor skyColors,
             MFString scenery ]
    groundColors []
    skyColors []
    scenery "maps/shspace.png"
}
```

Background Horizon Texture

The DSS HORIZON_ELEMENT keyword and its parameters are mapped to the DSpace_Horizon node and its fields with only one modification, which is the conversion of angle values from degrees to radians.

To convert degrees to radians, use this formula:

DSS HORIZON_ELEMENT: "http://anyserver.com/places/
maps/image.bmp", 180.0, 0.0, 5.0

```
VRML DSpace_Horizon {
  fields [ SFString url,
           SFFloat angularWidth,
           SFFloat bearing,
           SFFloat elevation ]
  url "maps/image.png"
  angularWidth 3.142
  bearing 0
  elevation 0.087
}
```

Light Sources

You can map the RGB value of the DSS DIRECTIONAL_LIGHT keyword to the intensity field of the DirectionalLight node and direction vector to its corresponding field.

DSS DIRECTIONAL_LIGHT: 0.8, 0.8, 0.8, 0, -1, 0

```
VRML DirectionalLight {
  intensity 0.8
  color 1 1 1
  direction 0 -1 0
  on TRUE
}
```

degrees () 180.0 () ÷ 3.142 × radians =

3D and 2D Objects

In this category, the COLLIDES DSS keyword is obsolete. Collision detection is automatically turned on by Digital Space Traveler for portals and links (and no other objects).

3D Geometry or Sprite Definitions

You can use the inlining technique to put your 3D object(s) in the VRML .dsv file. For sprites, the SPRITE_PROP DSS keyword and its parameters are easily mapped to the VRML DSpace_Sprite node and its fields without any changes, except that you can now use relative URLs.

DSS PROP:"object_name", "http://anyserver.com/places/
props/obj.3ds"

or

```
SPRITE_PROP: "object_name",
"http://anyserver.com/
places/sprites/image.bmp", 256, 0, 20
```

VRML WWWInline { name "props/obj.wrl" }

or

```
Dspace_Sprite {
fields [ SFString url,
         SFLong colorCount,
         SFFloat width,
         SFFloat height ]
url "sprites/image.png"
colorCount 256
width 0
height 20
}
```

Transformations

Due to differences in coordinate systems and units for measuring angles between VRML and DSS spaces, the translation z-value must be negated (MOVE_PROP is DSS).

Degrees in VRML are converted to radians for rotation. (To convert degrees to radians, use this formula: $\text{degrees} \times \frac{\pi}{180.0} = \text{radians}$) Also, *degrees* $(\text{degrees}) \div 180.0 \times \pi = \text{radians}$ rotation in VRML is set by four values versus three in DSS: x-axis, y-axis, z-axis, and rotation amount.

The following example shows a typical conversion of transformations from DSS to VRML:

DSS PROP:"object_name", "http://anyserver.com/places/props/obj.3ds"
MOVE_PROP: "object_name", 10, 5, -20
ROTATE_PROP: "object_name", 90, 0, 0
SCALE_PROP: "object_name", 0.15, 0.15, 0.15

or

```
SPRITE_PROP: "object_name",
"http://anyserver.com/
places/sprites/image.bmp", 256, 0, 20
MOVE_PROP: "object_name", 10.0, 5.0, 20.0
```

VRML Separator {
Transform {

```

        translation 10 5 20
        rotation 0 1 0 1.571
        scaleFactor 0.15 0.15 0.15
    }
    WWWInline { name "props/obj.wrl" }
}

```

or

```

Separator {
    Transform { translation 10 5 20 }
    DSpace_Sprite {
        fields [ SFString url,
                SFLong colorCount,
                SFFloat width,
                SFFloat height ]
        url "sprites/image.png"
        colorCount 256
        width 0
        height 20
    }
}

```

Spinning

Like rotations, the difference between DSS and VRML is that in the latter you set the spinning using 4 values instead of three: x-axis, y-axis, z-axis (spinning axes), the spinning speed value.

```

DSS PROP:"object_name", "http://anyserver.com/places/
      props/obj.3ds"
      SPIN_PROP: "object_name", 3, 0, 0

```

```

VRML Separator {
    DSpace_Spin {
        fields [ SFRotation rotation ]
        rotation 1 0 0 3
    }
    WWWInline { name "props/obj.wrl" }
}

```

Regular Portals and Links

In VRML spaces, you do not need to include the words 'portal' or 'link' in the name of that portal or link. Digital Space Traveler automatically detects portals or links and puts a prefix of either 'Portal:' or 'Link:', as appropriate, in front of the names on the pull-down menu. In the code, this looks like this:

```

DSS PROP:"portal - SPORTS", "http://anyserver.com/
      places/props/obj.3ds"
      PORTAL_DESTINATION: "portal - SPORTS", "http://

```

```
anyserver.com/places/sports.dss"
```

```
VRML WWWAnchor {
  name "sports.dsv"
  description "SPORTS"
  WWWInline { name "props/obj.wrl" }
}
```

User Portals

To create user portals, you need to follow the same procedure as regular portals, then add the `Dspace_UserPortal` node before the `WWWAnchor` node, as follows:

```
DSS PROP:"portal - USER SPACE",
      "http://anyserver.com/
      places/props/obj.3ds"
      USER_PORTAL: "portal - UserSpace", "http://
      anyserver.com/places/user.dss"
```

```
VRML Separator {
  Dspace_UserPortal {
    fields []
  }
  WWWAnchor {
    name "user.dsv"
    description "UserSpace"
    WWWInline { name "props/obj.wrl" }
  }
}
```

Shading Quality for 3D Objects

The default shading of 3D objects in the converted `.wrl` files is flat or faceted.

This means that if you do not specify a shading quality, objects' shading in the space will be flat.

If you want to change the shading quality to gouraud (smooth) shading for the whole `.wrl` file, use the `creaseAngle` field in the `ShapeHints` node.

Any `creaseAngle` value greater than 3.142 infers gouraud shading. If you want self-illuminant (unlit flat) objects, use the `emissiveColor` field in the `Material` node.

Both options are shown in the following example:

```
DSS PROP_QUALITY: "object_name", GOURAUD
```

or

```
PROP_QUALITY: "object_name", UNLIT_FLAT
```

VRML For *smooth (gouraud) shading*, add the following line at the top of the .wrl file (not the main .dsv file) after the root node:

```
Separator {
    ShapeHints { creaseAngle 4 }
    # rest of the file
}
```

For *unlit flat shaded objects*, edit the `Material` node in .wrl file to look like the following:

```
Material {
    ambientColor []
    diffuseColor []
    specularColor []
    emissiveColor [ 0.92157 0.83137 0.73333 ]
    shininess 0
    transparency 0
}
```

Global Change of Colors for 3D Objects

If you want to set the color of all objects in a .wrl file to a specific color, delete all `Material` nodes in the .wrl file and add a `Material` node before the inlined file in .dsv file. You need to put both nodes in a `Separator` node to constrain the effect of the `Material` node only to the subsequent node.

```
DSS PROP:"object_name", "http://anyserver.com/places/
props/obj.3ds"
COLOR_PROP: "object_name", 0.7, 0.1 , 0.1
```

```
VRML Separator {
    Material { diffuseColor 0.7 0.1 0.1 }
    WWWInline { name "props/obj.wrl" }
}
```

Prop Sounds

The main difference between DSS and VRML in this area is that in VRML, the sound is not directly associated with a 3D or 2D object. In VRML, sound can be defined as an independent object by the `DSPACE_Sound` node.

Because of the length of this particular node, it is recommended that you keep each sound source as a separate .wrl file, leaving the `location` field as 0 0 0.

You can then use the inlining technique to put this object in the main .dsv file, and use the `Translation` node to specify a particular position.

```
DSS PROP:"object_name", "http://anyserver.com/places/
```

```
props/obj.3ds"
PROP_SOUND: "object_name, "http://anyserver.com/
places/wavs/spound.wav", 1.0, 20.0
```

VRML The RAS tool creates a VRML .wrl file for the sound object (e.g., sound1.wrl), which should be filed in the .../props directory. This .wrl file's content will resemble the following:

```
#VRML V1.0 ascii
Separator {
  DSpace_Sound {
    # fields
  }
}
```

In addition, add the following lines in your main .dsv space file:

```
Separator {
  Translation { 10 5 20 }
  WWWInline { name "props/sound1.wrl" }
}
```

where 10 5 20 refer to the x, y, and z coordinates of a location in your space.

Ambient Audio

The AMBIENT_AUDIO keyword and its parameters are easily mapped to the DSpace_AmbientAudio node and its fields without any changes, *except* that the values of leftVolume and rightVolume fields in the VRML node are the products of the corresponding DSS parameters and the channel volume (the last parameter in the DSS keyword).

For example:

```
DSS AMBIENT_AUDIO: "http://anyserver.com/places/wavs/
aztec.wav", -1, 0, 0.00, 0.00, -1, 0, 0.00, 0.00,
0.50,0.50, 0.75
```

```
VRML DSpace_AmbientAudio {
  fields [ SFString url,
           SFLong innerLoopMinCount,
           SFLong innerLoopCountRange,
           SFFloat innerLoopMinDelay,
           SFFloat innerLoopDelayRange,
           SFLong outerLoopMinCount,
           SFLong outerLoopCountRange,
           SFFloat outerLoopMinDelay,
           SFFloat outerLoopDelayRange,
```

```

        SFFloat leftVolume,
        SFFloat rightVolume ]
url "wavs/aztec.wav"
innerLoopMinCount -1
innerLoopCountRange 0
innerLoopMinDelay 0.00
innerLoopDelayRange 0.00
outerLoopMinCount -1
outerLoopCountRange 0
outerLoopMinDelay 0.00
outerLoopDelayRange 0.00
leftVolume 0.38
rightVolume 0.38
}

```

Adding DSpace_EntryPoint Node

You can specify where avatars start out when they enter the space by adding one or more `DSpace_EntryPoint` nodes. If more than one node is used, the entry point is picked at random by Traveler. Refer to section A.2 for more details on this node.

```

VRML DSpace_EntryPoint {
    fields [ SFVec3f position,
            SFRotation rotation,
            SFFloat width,
            SFFloat height,
            SFFloat depth,
            SFBool fly,
            SFVec3f startPosition,
            SFRotation startRotation ]
    position 0 0 0
    rotation 0 1 0 0
    width 1
    height 1
    depth 0
    fly TRUE
    startPosition 0 10 0
    startRotation 1 0 0 -1.571
}

```

B.3 DSS Specification

B.3.1 Overview

A virtual space is described by a Space Description Script (DSS). An DSS specifies general and environmental properties of a space, including a list of objects such as 3D geometry, 2D sprites, audio elements, and lights which the designer populates a space with. A space designer also modifies or sets the properties of these objects and applies transformations to them by using DSS keywords.

B.3.2 File Format

An DSS is an ASCII text file consisting of a header, a list of DSS keywords and corresponding parameters, and comments. Each element occupies exactly one line of text and generally consists of a keyword followed by a colon, one or more parameters, and an optional semi-colon and following comment. Values for keyword parameters can be floating-point numbers, integers, strings, keyword-specific types, or comma-separated, ordered sets of primitive types, depending on the keyword. Empty lines, white spaces (outside of quotes), lines beginning with unrecognized keywords, misformatted lines, comments, data preceding a valid header or any data after the "END" keyword are all ignored. All keywords are case-insensitive. All space description elements are optional. Omitting a keyword causes a default value to be assumed. (i.e., an DSS consisting only of a header defines a valid space, described entirely by default values.) The list of space description elements is not order-dependent. Duplicating the use of a keyword causes the value provided with the last instance to completely replace the values provided with any previous instances.

B.3.3 Data

DSS supports inclusion of binary data as resources for 3D geometry, bitmaps, and audio. The Digital Space Traveler 2.0 supports the following file formats:

3D Geometry

3D objects, defined by the `PROP` keyword (e.g., Autodesk 3D Studio .3ds file format).

Bitmaps

There are four elements in DSS spaces which use bitmaps: 2D sprites, sky tiles, horizon maps, and texture maps. They all use Windows BMP file format, both uncompressed and RLE compressed (for 8-bit files).

However, there are special requirements for each.

2D sprite objects, defined by the `SPRITE_PROP` keyword, use both 8-bit and 24-bit .bmp files. Digital Space recommends that you use only 8-bit files since Digital Space Traveler uses the ramp color model, meaning that Traveler only uses 256 colors. You can make some areas of a sprite transparent by filling those areas with color black (RGB = 0, 0, 0), assuming that you *do not* have any other palette entries with the color black. By having more than one palette entry as black, you can have opaque black in your sprite. With transparent black, the closest opaque color to black is dark gray (RGB = 10, 10, 10). There are no special requirements for the pixel dimensions of the sprites.

Sky tiles and horizon maps, which are part of the environmental properties of the spaces and defined by the `SKY_TILE` and `HORIZON_ELEMENT` keywords, use 8-bit .bmp files as their data resources. They do not contain any transparency information. Their width and height dimensions should be a value of 2n. For example, 4 pixels by 4 pixels, or 512 pixels by 256 pixels.

Texture maps are not directly referred to in DSS. The pointer to the name of a .bmp file used as a texture map is saved in the corresponding .3ds file. However you should specify the directory path of the files using the `BMP_PATH` keyword. Texture maps files should have upper-cased filenames, so as to be compatible with case-sensitive operating systems such as UNIX. Texture maps also use 8-bit .bmp files as their data resources. Their width and height dimensions should be a value of 2n. For example, 8 pixels by 8 pixels, or 64 pixels by 128 pixels.

Transparency for texture maps is applied in 3D Studio, by using a separate .bmp file (with the same specifications as the texture map) assigned to a `Material` as an opacity map.

Audio

All classifications of Digital Space Traveler audio events use the Microsoft .wav file format in either 8- or 16-bit depths. Sample rates must be 8 kHz (optimal) or 16 kHz to meet the requirements of a SoundBlaster full-duplex driver.

The drivers also support stereo files.

The audio driver supports a compression routine using the OLW file format. Using the Digital Space RAS audio sequencing tool, .wav files can be compressed (15:1 lossy) into .olw files for download purposes. The compressed files are then expanded back to a .wav file upon client

playback. The source specifications of the pre-compressed files are 16 kHz sample rate and 16-bit sample depth.

B.3.4 .dss File Extension

The file extension for DSS space files is .dss.

The following sections provide more details on an DSS header and keywords used in the DSS specification.

B.3.5 Header

No information will be processed until a valid header line is encountered in an DSS file. Typically, the header is the first line in an DSS file (although it could be preceded by comments). Following is the syntax of a typical DSS header:

```
<script_type, version, "access_server", "title",  
max_avatars,max_instances>
```

Where:

- `script_type` refers to the type of the file being read. This is always DSS.
- `version` is the version of the application for which the script was written. This should be 1.0 until an extended spec is published and extended features are used.
- `"access_server"` includes the name of the Digital Space access server.
- `"title"` is a string specifying the name, or title, of the space. Only the first 32 characters are significant (i.e., all characters after the 32nd are ignored). Any character is allowed in the name, which is delimited by double straight-quotes (" "). The standard ANSI escape character conventions will be recognized for including quotes or forward slashes (/) in the string. Note that while this spec allows any character in a space's name, the Digital Space application may be more restrictive about what constitutes a legal name.
- `max_avatars` is an integer value specifying the maximum number of avatars allowed in every instance of the space. If this value is 0, then the determination of maximum number of avatars is made by the access server. For user created rooms, this number may be decreased, but not increased.
- `max_instances` is an integer value specifying the maximum number

of instances, or clones, of the space that will be allowed (created) on an access server. If this value is 0, then the determination of maximum number of clones is made by the access server.

The following are examples of valid DSS header lines:

Example 1: <DSS, 1.0, " anyserver.com", "Utopia Gateway", 12, 0>

Example 2: <DSS, 1.0, " anyserver.com", "Lobby//Foyer", 0, 0>

; note, the name (title) is this example is
Lobby/Foyer

Example 3: <DSS, 1.0, " anyserver.com", "/"Kids/" Only", 0, 0>

; note, the name (title) in this example is: "Kids"
Only

B.3.6 DSS Keywords

Following is an alphabetic listing of each keyword in the DSS specification.

AMBIENT_AUDIO

```
AMBIENT_AUDIO: "url", innerLoopMinCount,  
innerLoopCountRange, innerLoopMinDelay,  
innerLoopDelayRange, outerLoopMinCount,  
outerLoopCountRange, outerLoopMinDelay,  
outerLoopDelayRange, leftVolume, rightVolume,  
channelVolume
```

AMBIENT_AUDIO adds an ambient sound to the space. `url` is a quoted delimited string naming the sound resource to add. `group` is a nonnegative integer indicating a sequence group for this sound resource.

When a sound resource finishes being played the next resource in the sequence group is started. When the last resource in a sequence group finishes being played and the sequence group includes more than one ambient sound, the whole group is repeated. If there is only one item in a sequence group, then continuous looping is accomplished by setting the `minLoops` and `maxLoops` parameters to -1.

`minLoops` and `maxLoops` are integers specifying the minimum and maximum number of loops of the sound resource to play before continuing to the next ambient sound. If the two values are different, a random value between them is chosen each time through the sequence

group. To cause a sound resource to loop continuously, set both values to -1. `minDelay` and `maxDelay` are non-negative floating point values specifying the minimum and maximum number of seconds to pause between loops of the sound resource. If the two values are different, a random delay between the values chosen between loops. There is no pause before the first loop or after the last loop of the sound resource.

AMBIENT_LIGHT

AMBIENT_LIGHT: red, green, blue

Ambient light source illuminates everything in the space, irrespective of the orientation, position, and surface properties of the objects in the space.

This keyword sets the single ambient light level, replacing the previous ambient light if one was set. The floating point values given for the red, green and blue light components range from 0 to 1. The current version of the Digital Space Traveler supports ramp color model (8-bit or 256 colors).

This color model ignores the color content of the light and uses the gray component of each light to calculate a single gray intensity. The intensity of the light is based on the following equation:

$$\text{intensity} = 5/16*\text{red} + 9/16*\text{green} + 2/16*\text{blue}$$

BACKGROUND_COLORS

BACKGROUND_COLORS: number

An integer value specifying how many colors out of the 8-bit palette should be set aside for the sky and ground colors, the sky tile and all of the horizon elements. This is valid only for 8-bit rendering models.

BMP_PATH

BMP_PATH: "url"

Specify http path (url) of where the texture map image files (BMP) can be found. Texture map images are bound to a 3D object via 3D Studio material editor. DSS ignore the texture path that 3DS binds to the object and only uses the actual image filename. The `BMP_PATH` then tell DSS where to find the texture image files. Texture image files *must* be in upper-case for the process to work.

BOUNDING_BOX

BOUNDING_BOX: minX, minY, minZ, maxX, maxY, maxZ

Specifies the bounding box for the space. The bounding box is axis

aligned with diagonally opposite corners at the points (minX, minY, minZ) and (maxX, maxY, maxZ).

COLLIDES

COLLIDES: "object_name", "ON" | "OFF"

Toggles collision detection for individual objects.

COLLISION

COLLISION: ON|OFF

Toggles collision detection for the whole space.

COLOR_PROP

COLOR_PROP: "object_name", red, green, blue

Colors all faces of the named prop with the given color. red, green, blue components are floating point numbers between 0 and 1.

DIRECTIONAL_LIGHT

DIRECTIONAL_LIGHT: red, green, blue, vectorX,
vectorY,
vectorZ

A directional light source has direction but no position. It illuminates all objects in a space with equal intensity, as if it were at an infinite distance from each object. This keyword creates a new directional light of the intensity specified by red, green, and blue. The light points in the direction specified by the floating point values vectorX, vectorY, and vectorZ. The floating point values given for the red, green and blue light components range from 0 to 1. The current version of the Digital Space Traveler supports ramp color model (8-bit or 256 colors). This color model ignores the color content of the light and uses the gray component of each light to calculate a single gray intensity. The intensity of the light is based on the following equation:

$$\text{intensity} = 5/16*\text{red} + 9/16*\text{green} + 2/16*\text{blue}$$

Directional lights are additive.

END

END:

The `END` keyword optionally defines the point in the file, after which no more keywords will be processed. An DSS reader will also recognize the end of the file as a legal conclusion to the space description. This keyword is primarily intended to be used in tokenized streams, rather than in human-readable DSS's, although a space-description could conclude with some explanatory comments after the `END` keyword.

FIELD_OF_VIEW

`FIELD_OF_VIEW`: degrees

Specifies the degrees of horizon displayed in the view window. The default field of view for a space is 90 degrees.

GROUND_COLOR

`GROUND_COLOR`: red, green, blue

This is the color of the ground in a half space (or any other future category where a ground color makes sense). The same comments apply to the red, green, blue components (floating point numbers between 0 and 1) as to the components of the `SKY_COLOR` above.

The default is a dark, grassy green (RGB = 0.2, 0.6, 0.2).

HOME_ADDRESS

`HOME_ADDRESS`: "url"

Specifies the url for the current .dss file.

HORIZON_ELEMENT

`HORIZON_ELEMENT`: "url", angular_width, bearing, elevation

url is the name of the BMP resource containing the image data for a horizon element (a feature on the horizon at an infinite distance from the user. Like the image data for the `SKY_TILE`, the width and height dimensions of a `HORIZON_ELEMENT` should be a value of 2^n (although not necessarily the same power of two). The angular-width parameter is a floating point value, specifying the angle in degrees of horizon that the `HORIZON_ELEMENT` occupies. For example, a value of 180.0 for angular width, specifies that an element takes up half the horizon (from due East to due West, for example). The bearing is a floating point number specifying an angle (in degrees) at which to place the lower left edge of the horizon element. The elevation indicates in degrees how "high" above the horizon the element should be. It should

range between -90.0 and 90.0. A value of 0 means the bottom of the element is resting on the virtual horizon. A value of -45.0 would specify that a user would have to tilt their virtual head back 45.0 degrees to center the element vertically in their view.

Unlike most keywords, this one is cumulative in effect. That is, a space can have an arbitrary number of horizon elements. At run-time, the elements are processed in the order they are specified, from back to front. Hence, the order of specification defines a priority order for displaying multiple horizon elements. The default state for a space is no horizon elements.

INCLUDE

```
INCLUDE: "url"
```

Specifies a resource that will be processed before continuing processing the current DSS. All DSS keywords in the included resource are interpreted as if they replace the `INCLUDE` keyword itself.

MOVE_PROP

```
MOVE_PROP: "object_name", deltaX, deltaY,  
           deltaZ
```

Moves the prop in the 3D space by distance specified in the three spatial components: `deltaX`, `deltaY`, and `deltaZ` (floating point values). By default, all props are added at absolute (0.0, 0.0, 0.0). Since absolute positions are meaningless in full and half-spaces, this is mostly useful in positioning the props relative to one another.

PORTAL_DESTINATION

```
PORTAL_DESTINATION: "object_name", "url"
```

Specifies the url to traverse when an avatar collides with the prop. This `url` should reference a resource that represents a destination. Unless the browser itself can display this resource, it will minimize itself after invoking an appropriate handler. By default there is no collision destination for a prop.

PROP

```
PROP: "object_name", "url"
```

Adds a prop to a space. `url` is a quote-delimited string with the name of the resource in which the prop is defined. Currently, the only type

supported is a .3ds file. `object_name` is another string which is the name used to refer to the prop in prop-modification keywords. By default there are no props in a space. As in the case of `HORIZON_ELEMENT`, `PROP` is a cumulative keyword, adding a new prop each time it is used.

PROP_QUALITY

```
PROP_QUALITY: "object_name",  
  
WIRE_FRAME | UNLIT_FLAT | FLAT | GOURAUD
```

Assigns a rendering quality to the specified prompt. Prop-name is the name of the prop. Quality can take on the following values: `WIRE_FRAME`, `UNLIT_FLAT`, `FLAT`, or `GOURAUD`. The default value is `GOURAUD`.

PROP_SOUND

```
PROP_SOUND: "object_name", "url", volume,  
            attenuation_circumference
```

Specifies the url to the sound resource to be attached to the indicated prop. This sound resource will be panned and attenuated so as to sound like it is coming from the prop based on the size of the attenuation circumference. `volume` is a floating point number between 0.0 and 1.0 that specifies the volume for the sound. By default there is no sound for a prop.

ROTATE_PROP

```
ROTATE_PROP: "object_name", yaw, pitch, roll
```

Rotates a prop by the specified angles (floating point values, in degrees). In a coordinate space defined by the prop, `yaw` represents rotation about a vertical axis, `pitch` is rotation about the "right" axis and `roll` is rotation about the "forward" axis. By default, a prop is added in whatever orientation is implied by the .3ds file that defines it.

SCALE_PROP

```
SCALE_PROP: "object_name", scaleX, scaleY, scaleZ
```

Scales a prop along each of 3 perpendicular axes. `scaleX`, `scaleY` and `scaleZ` are scale factors. A value of 1.0 means no scaling should occur along an axis. A value of 0.5 indicates a reduction to 50% of the dimension's original size. A value of 2.0 indicates an expansion to 200%

of the dimension's original size.

Using identical values for `scaleX`, `scaleY`, and `scaleZ` will cause the prop to be proportionally shrunk or grown by a certain amount.

Through trial and error, a designer can determine the exact effect of scaling independently along the three axes. The intended use for this keyword is to allow a scene designer to instantiate multiple props from a single `.3ds` file, while sizing the different instances independently. Due to numeric loss of significance, scaling a prop multiple times in the same script could cause accumulating errors that would significantly change the geometry of the prop.

SKY_COLOR

```
SKY_COLOR: red, green, blue
```

Specifies the overall background color of the space. In a full space, this is the color the user sees in all directions behind objects and other avatars. In a half space, this is the color the user sees in the "sky" above the "ground". This color will not be seen if a sky-tile is specified, but remains part of the description of the space in the event that the sky-tile is removed.

Red, blue and green are the RGB components of the color. Each component is a floating point number ranging from 0 to 1 (inclusive). Numbers in excess of 1 are truncated to 1 and numbers less than 0 are truncated to 0.

The default is black (RGB = 0.0, 0.0, 0.0).

SKY_TILE

```
SKY_TILE: "url"
```

This is a quote-delimited string with the name of the BMP resource containing the pattern to be replicated across the sky in a virtually infinite spherical shell. The width and height dimensions of a `SKY_TILE` should be a value of 2^n (although not necessarily the same power of two). By default, there is no sky-tile.

SPIN_PROP

```
SPIN_PROP: "object_name", spinX, spinY, spinZ
```

Specifies a constant rotational speed around the center of the object, set in the `.3ds` file. The rotation values are floating point numbers specifying

the rotations around that axis per second. By default a prop has no spin around any axis.

SPRITE_PROP

```
SPRITE_PROP: "object_name", "url", num_colors,  
width,height
```

Adds a sprite to a space. A sprite is a 2D bitmap representing a 3D object. It is always aligned with the camera, and is therefore best suited to displaying spherically or cylindrically symmetric objects. `object_name` is the name of the prop for reference by keywords, such as `MOVE_PROP`. `url` is a quote-delimited string specifying the path to the .bmp file that serves as the source for the sprite. `num_colors` specifies how many of the 256 palette entries the sprite will use when in view. `width` and `height` are floating-point values representing the width and height of the object in the 3D environment. Specifying a width or height of 0.0 indicates that that dimension should be derived from the natural aspect ratio of the source image. Specifying a width and height of 0.0 indicates that the object does not scale with distance at all.

TYPE

```
TYPE: FULL|HALF
```

Specifies the major category of the space. A half space has a floor which exerts planar pseudo-gravity while full spaces are "groundless" spaces with objects with optional spherical pseudo-gravity. The default is `FULL`.

USER_PORTAL

```
USER_PORTAL: "object_name", "url"
```

Specifies the `url` to traverse when an avatar collides with the prop. This `url` should reference a resource that represents a destination. Unless the browser itself can display this resource, it will minimize itself after invoking an appropriate handler. By default there is no collision destination for a prop.

B.3.7 DSS Example

Following is a commented .dss file example to illustrate the use of DSS keywords.

```
<DSS, 1.0 , " anyserver.com", "Space Name", 12, 0> ; required header
; Copyright © Digital Space Technologies, 1996, all rights reserved.
; Information about designers
HOME_ADDRESS: "http://anyserver.com/places/sample.dss" ; required
BMP_PATH: "http://anyserver.com/places/texmaps"
; space environmental properties
; avatars will not be able to move through the floor (y=0)
BOUNDING_BOX: -1000.0, 0.0, -1000.0, 1000.0, 1000.0, 1000.0
COLLISION: ON ; space will have some collision elements
FIELD_OF_VIEW: 90.0 ; camera view of avatar
AMBIENT_LIGHT: 0.2, 0.2, 0.2 ; ambient lighting of overall scene
TYPE: FULL ; no ground floor, sky tile wraps through-out space
; number of colors in the 256 palette to reserve for the bitmaps(sky and
horizon)
BACKGROUND_COLORS: 44
; image to wrap on infinite sphere ( except where horizon map is)
SKY_TILE:"http://anyserver.com/artopia/ftikki/maps/bitmap1.bmp"
; images to wrap horizontally around horizon which is 360 degrees
; notice images are 90 degrees each and start at 90 degree intervals
HORIZON_ELEMENT: "http://anyserver.com/places/maps/bitmap2.bmp", 90.0,
0.0, 15.0
HORIZON_ELEMENT: "http://anyserver.com/places/maps/bitmap3.bmp", 90.0,
90.0, 15.0
HORIZON_ELEMENT: "http://anyserver.com/places/maps/bitmap3.bmp", 90.0,
180.0, 15.0
HORIZON_ELEMENT: "http://anyserver.com/places/maps/bitmap3.bmp", 90.0,
270.0, 15.0
; 4 infinite directional lights with r, g, b and x, y, z of light vector
whose base is at local 0,0,0
DIRECTIONAL_LIGHT: 0.9, 0.9, 0.9, 1.0, -1.0, 1.0
DIRECTIONAL_LIGHT: 0.1, 0.1, 0.1, -1.0, -1.0, 1.0
DIRECTIONAL_LIGHT: 0.2, 0.2, 0.2, 0.0, 0.5, -1.0
DIRECTIONAL_LIGHT: 0.6, 0.6, 0.6, -1.0, 0.1, 0.0
; calls 1 of 2 3DS files that make up main structure and renders it with
flat shading
; flat shaded object
PROP: "object name 1", "http://anyserver.com/places/props/object1.3ds"
PROP_QUALITY: "object name 1", FLAT
COLLIDES: "object name 1", "OFF"
; calls 2 of 2 3DS files that make up main structure and renders it with
unlit_flat shading
; self-illuminated object
PROP: "object name 2", "http://anyserver.com/places/props/object2.3ds"
PROP_QUALITY: "object name 2", UNLIT_FLAT
COLLIDES: "object name 2", "OFF"
; texture mapped object
PROP: "object name 3", "http://anyserver.com/places/props/object3.3ds"
PROP_QUALITY: "object name 3", UNLIT_FLAT ; texture mapped objects
generally use UNLIT_FLAT
MOVE_PROP: "object name 3", 101.10, -1.5, -154.14
SCALE_PROP: "object name 3", 1.0, 3.3, 3.0
SPIN_PROP: "object name 3", 3.2, 0.0, 0.0
COLLIDES: "object name 3", "OFF"
; spinning star
```



```
PROP: "star", "http://anyserver.com/places/props/star.3ds" ; adds a 3D
object to the space
PROP_QUALITY: "star", FLAT ; sets object's shading model to FLAT
COLOR_PROP: "star", 0.0, 0.0, 1.0 ; overwrites object's default colors
with color blue
MOVE_PROP: "star", -42.0, 15.0, -25.0 ; moves object
SPIN_PROP: "star", 1.5, 0.0, 1.5 ; spins object 1.5 rotations per second
in x and z
COLLIDES: "star", "OFF" ; toggles object's collision detection off
; sprite
SPRITE_PROP: "sprite 1", "http://anyserver.com/places/sprites/
spritel.bmp", 256, 0.0, 50.0
MOVE_PROP: "sprite 1", 0.0, -90.0, -149.0
COLLIDES: "sprite 1", "OFF"
PROP_SOUND: "sprite 1", "http://anyserver.com/places/wavs/sound1.olw",
1.0, 150.0
; portals and links
; link - Digital Space Page
SPRITE_PROP: "link - DSpace Page ",
"http://anyserver.com/places/sprites/
sprite2.bmp", 256, 0.0, 20.0
MOVE_PROP: "link - Digital Space Page ", 0.0, -40.0, -215.0
COLLIDES: "link - Digital Space Page ", "ON"
PORTAL_DESTINATION: "link - Digital Space Page ", "http://
http://www.digitalspace.com /"
PROP_SOUND: "link - Digital Space Page ",
"http://anyserver.com/places/wavs/
webpage.olw", 1.0, 30.0
; portal - Another Space
PROP: "portal - Another Space", "http://anyserver.com/places/props/
portall.3ds"
PROP_QUALITY: "portal - Another Space ", FLAT
COLLIDES: "portal - Another Space ", "ON"
MOVE_PROP: "portal - Another Space ", 150.0, 72.0, -147.9
PORTAL_DESTINATION: "portal - Another Space ", "http://anyserver.com/
places/space2.dss"
; portal - User Space
PROP: "portal - User Space",
"http://anyserver.com/places/props/user1.3ds"
PROP_QUALITY: "portal - User Space", FLAT
COLLIDES: "portal - User Space", "ON"
MOVE_PROP: "portal - User Space", 169.7, 72.0, -147.9
USER_PORTAL: "portal - User Space",
"http://anyserver.com/places/user.dss"
; two ambient audio sequences, one repeats forever
AMBIENT_AUDIO: "http://anyserver.com/places/wavs/j2.olw", -1, 0, 0, 0, -
1, 0, 0, 0, 0.5, 0.5, 0.75
AMBIENT_AUDIO: "http://anyserver.com/places/wavs/jungst.olw", 1, 0, 0,
0, 1, 0, 0, 0, 0.5, 0.5, 1
END: ; no keywords will be processed after this line
```


C • • • Appendix

C.1 VRML Quick Reference

VRML Ver. #	Node	Field	Default Value	Value Type	Value Range	Traveler Implemented
1	(header)	<i>Every VRML file must begin with the characters: #VRML V1.0 ascii</i>				Full Support
1	AsciiText	<i>Represents strings of text characters.</i>				Full Support
		string	""	MFString	any	
		spacing	1	SFFloat	any	
		justification	Left	SFEnum	LEFT CENTER RIGHT	
		width	0	MFFloat		
1	Cone	<i>Represents a cone shape.</i>				Full Support
		parts	ALL	SFBitMask	SIDES BOTTOM ALL	
		bottomRad	1	SFFloat	>0	
		height	2	SFFloat	>0	
1	Coordinate3	<i>Defines a set of 3D coordinates for subsequent vertex-based shape nodes</i>				Full Support
		point	0 0 0	MFVec3f	any	
1	Cube	<i>Represents a cube shape.</i>				Full Support
		width	2	SFFloat	>0	
		height	2	SFFloat	>0	
		depth	2	SFFloat	>0	
1	Cylinder	<i>Represents a cylinder shape.</i>				Full Support
		parts	ALL	SFBitMask	SIDES TOP BOTTOM ALL	
		radius	1	SFFloat	>0	
		height	2	SFFloat	>0	
1	DEF	<i>Keyword that is used for instancing, gives a node a name and creates copy of the node</i>				Full Support
1	DirectionalLight	<i>Defines a directional light source.</i>				Full Support
		on	TRUE	SFBool	TRUE FALSE	
		intensity	1	SFFloat	0-1	
		color	1 1 1	SFColor	0-1	
		family	SERIF	SFEnum	SERIF SANS TYPEWRITER	
		style	NONE	SFBitMask	NONE BOLD ITALIC	

VRML Quick Reference (continued)

VRML Ver. #	Node	Field	Default Value	Value Type	Value Range	Traveler Implemented
1	Group	<i>Defines the base class for all group nodes (deprecated).</i>				Full Support
1	IndexedFaceSet	<i>Constructs a 3D shape by drawing its faces from current indexed list of vertices.</i>				Full Support
		coordIndex	0	MFLong	-1 or > = 0	
		materialIndex	-1	MFLong	-1 or > = 0	
		normalIndex	-1	MFLong	-1 or > = 0	
		textrueCoordIndex	-1	MFLong	-1 or > = 0	
1	IndexedLineSet	<i>Constructs a 3D polyline shape from current indexed list of vertices.</i>				Not Supported
		coordIndex	0	MFLong	-1 or > = 0	
		materialIndex	-1	MFLong	-1 or > = 0	
		normalIndex	-1	MFLong	-1 or > = 0	
		textureCoordIndes	-1	MFLong	-1 or > = 0	
1	Info	Contains any information text string.				Full Support
		string	"<Undefined info>"	SFString	any	
1	LOD	<i>Group node that allows switching between various representations of objects</i>				Partial Support
		range	1	MFFloat		
		center	0 0 0	SFVec3f		
1	Material	<i>Defines surface material properties for subsequent shapes</i>				Partial support.
		ambientColor	0.2 0.2 0.2	MFCColor	0-1	Ignored
		diffuseColor	0.8 0.8 0.8	MFCColor	0-1	Standard.
		specularColor	0 0 0	MFCColor	0-1	Ignored.
		emissiveColor	0 0 0	MFCColor	0-1	if emissiveColor > diffuseColor, then material is self-illuminant
		shininess	0.2	MFFloat	0-1	Ignored.
		transparency	0	MFFloat	0-1	Ignored.

VRML Quick Reference (continued)

VRML Ver. #	Node	Field	Default Value	Value Type	Value Range	Traveler Implemented
1	MaterialBinding	<i>Specifies how materials are bound to subsequent shapes</i>				Partial Support.
		value	OVERALL	SFEnum	DEFAULT OVERALL PER_PART PER_PART_INDEXED PER_FACE PER_FACE_INDEXED PER_VERTEX PER_VERTEX_INDEXED	PER_VERTEX maps to PER_FACE
1	Matrix Transform	<i>Specifies a 3D geometric transformation as a 4 by 4 matrix</i>				Full Support
		matrix	"1 0 0 0 0 1 0 0 0 0 1 0 0 0 0 1"	SFMatrix	any non-singular matrix	
1	Normal	<i>Defines a set of 3D surface normal vectors for subsequent vertex-based shape nodes.</i>				Full Support
		vector	[]	MFVec3f	any unit vector	
1	Normal Binding	<i>Specifies how surface normals are bound to subsequent shapes</i>				Partial Support
		value	DEFAULT	SFEnum	DEFAULT OVERALL PER_PART PER_PART_INDEXED PER_FACE PER_FACE_INDEXED PER_VERTEX PER_VERTEX_INDEXED	all but PER_PART and PER_FACE

VRML Quick Reference (continued)

VRML Ver. #	Node	Field	Default Value	Value Type	Value Range	Traveler Implemented	
1	OrthographicCamera	<i>Defines an orthographic camera</i>				no support	
		position	0 0 1	SFVec3f	any		
		orientation	0 0 1 0	SFRotation	any		
		focal Distance	5	SFFloat	>0		
		height	2	SFFloat	>0		
1	PerspectiveCamera	<i>Defines a perspective camera.</i>				Partial Support	
		position	0 0 1	SFVec3f	any		Standard.
		orientation	0 0 1 0	SFRotation	any		Standard.
		focal Distance	5	SFFloat	any		Ignored.
		height Angle	0.785398	SFFloat	>0,<PI		Ignored.
1	PointLight	<i>Defines a point light source.</i>				Full Support.	
		on	TRUE	SFBool	TRUE FALSE		
		intensity	1	SFFloat	0-1		
		color	1 1 1	SFColor	0-1		
		location	0 0 1	SFVec3f	any		
1	PointSet	<i>Creates a set of points from the current indexed list of coordinates.</i>				no support	
		startIndex	0	SFLong	>=0		
		numPoints	-1	SFLong	-1 or >=0		
1	Rotation	<i>Specifies a 3D rotation about an arbitrary axis through the origin.</i>				Full Support.	
		rotation	0 0 1 0	SFRotation	any		
1	Scale	<i>Specifies a 3D scaling about the origin</i>				Full Support.	
		scaleFactor	1 1 1	SFVec3f	>0		
1	Separator	<i>Group node that saves and restores traversal state</i>				Full Support.	
		renderCulling	AUTO	SFEnum	ON OFF AUTO		

VRML Quick Reference (continued)

VRML Ver. #	Node	Field	Default Value	Value Type	Value Range	Traveler Implemented
1	ShapeHints	<i>Provides hints about subsequent shapes</i>				Partial Support
		vertexOrdering	UNKNOWN_ORDERING	SFEnum	UNKNOWN_ORDERING CLOCKWISE COUNTERCLOCKWISE	default is CLOCKWISE
		shapeType	UNKNOWN_SHAPE_TYPE	SFEnum	UNKNOWN_SHAPE_TYPE SOLID	default is SOLID
		faceType	CONVEX	SFEnum	UNKNOWN_FACE_TYPE CONVEX	Standard.
		creaseAngle	0.5	SFFloat	any	default is 0.0, either <= or >=PI (for smooth shading)
1	Sphere	<i>Represents a sphere shape.</i>				Full Support.
		radius	1	SFFloat	>0	
1	SpotLight	<i>Defines a spotlight source.</i>				Full Support.
		on	TRUE	SFBool	TRUE FALSE	
		intensity	1	SFFloat	0-1	
		color	1 1 1	SFColor	0-1	
		location	0 0 1	SFVec3f	any	
		direction	0 0 -1	SFVec3f	any unit vector	
		dropOffRate	0	SFFloat	0-1	
		cutOffAngle	0.785398	SFFloat	0-PI	
1	Switch	<i>Group node that traverses one chosen, all, or none of its children</i>				Full Support.
		whichChild	-1	SFLong	>=0 -3 -3	
1	Texture2	<i>Defines a texture map for subsequent shapes.</i>				Partial Support
		filename	""	SFString	any	PNG, JPEG, and BMP supported
		image	0 0 0	SFImage	any	Ignored.
		wrapS	REPEAT	SFEnum	REPEAT CLAMP	
		wrapT	REPEAT	SFEnum	REPEAT CLAMP	

VRML Quick Reference (continued)

VRML Ver. #	Node	Field	Default Value	Value Type	Value Range	Traveler Implemented	
1	Texture2Transform	<i>Specifies a 2D transformation applied to texture coordinates of subsequent shapes.</i>				Full Support.	
		translation	0 0	SFVec3f	any		
		rotation	0	SFFloat	any		
		scaleFactor	1 1	SFVec3f	>0		
1	Transform	<i>Specifies a 3D geometric transformation</i>				Full Support.	
		translation	0 0 0	SFVec3f	any		
		rotation	0 0 1 0	SFRotation	any		
		scaleFactor	1 1 1	SFVec3f	>0		
		scaleOrientation	0 0 1 0	SFRotation	any		
center	0 0 0	SFVec3f	any				
1	TransformSeparator	<i>Group node that saves and restores transformation state (deprecated)</i>				Full Support.	
1	Translation	<i>Specifies a 3D geometric translation</i>				Full Support.	
		translation	0 0 0	SFVec3f	any		
1	USE	<i>Keyword that is used for instancing; indicates that a named node should be used again</i>				Full Support.	
1	WWWAnchor	<i>Group node that loads a new world when one of its children is chosen</i>				Partial Support	
		name	""	SFString			Standard.
		description	""	SFString			Standard.
		map	NONE	SFEnum	POINT NONE		Ignored.
1	WWWInline	<i>Group node that reads its children from anywhere in the World Wide Web</i>				Partial Support	
		name	""	SFString			Standard.
		bboxSize	0 0 0	SFVec3f			Ignored.
		bboxCenter	0 0 0	SFVec3f			Ignored.
1.0 + DS	DSPACE_AmbientAudio	<i>Defines an ambient sound source.</i>				Full Support.	
url	""	SFString					
1.0 + DS	DSPACE_AmbientAudio	<i>Defines an ambient sound source.</i>				Full Support.	
		url	""	SFString			
		innerLoopMinCount	-1	SFLong			
		innerLoopCountRange	0	SFLong			
		innerLoopMinDelay	0	SFFloat			
		innerLoopDelayRange	0	SFFloat			
		outerLoopMinCount	-1	SFLong			
		outerLoopCountRange	0	SFLong			
		outerLoopMinDelay	0	SFFloat			
		outerLoopDelayRange	0	SFFloat			
		leftVolume	1	SFFloat			
rightVolume	1	SFFloat					
1.0 + DS	DSPACE_CubeBoundary	<i>Defines a box that bounds avatar's movement in space</i>				Full Support	
		min	0 0 0	SFVec3f			
		max	1 1 1	SFVec3f			

VRML Quick Reference (continued)

VRML Ver. #	Node	Field	Default Value	Value Type	Value Range	Traveler Implemented
1.0 + DS	DSpace_EntryPoint	<i>Defines attributes of avatars's entry to a space, such as landing position and orientation</i>				Full Support
		position	0 0 0	SFVec3f		
		rotation	0 1 0 0	SFRotation		
		width	1	SFFloat		
		height	1	SFFloat		
		depth	0	SFFloat		
		fly	TRUE	SFBool	TRUE FALSE	
		startPosition	0 1 0 0	SFVec3f		
		startRotation	1 0 0 -1.5	SFRotation		
1.0 + DS	DSpace_Horizon	<i>Specifies a background texture at infinite distance.</i>				Full Support.
		url	""	SFString		
		angularWidth	3.142	SFFloat		
		bearing	0	SFFloat		
		elevation	0	SFFloat		
1.0 + DS	DSpace_MaxAvatars	<i>Specifies maximum number of avatars allowed in an instance of a space</i>				Full Support
		max	0	SFLong		
1.0 + DS	DSpace_MaxInstances	<i>Specifies maximum number of instances or clones of a space that can be created in a community.</i>				Full Support
		max	0	SFLong		
1.0 + DS	DSpace_Server	<i>Specifies name of Digital Space Access Server which the space is connected to.</i>				Full Support
		server	""	SFString		
		url	""	SFString		
1.0 + DS	DSpace_Sound	<i>Defines a sound source.</i>				Full Support
		url	""	SFString		
		location	0 0 0	SFVec3f		
		direction	0 0 1	SFVec3f		
		maxBack	1	SFFloat		
		maxFront	1	SFFloat		
		minBack	0	SFFloat		
		minFront	0	SFFloat		
		intensity	1	SFFloat		
		priority	0.5	SFFloat		
		spatialize	0	SFLong		
		innerLoopsMin	1	SFLong		
		innerLoopsRange	0	SFLong		
		innerLoopsDelayMin	0	SFFloat		
		innerLoopsDelayRange	0	SFFloat		
		outerLoopsMin	1	SFLong		
		outerLoopsRange	0	SFLong		
		outerLoopsDelayMin	0	SFFloat		
outerLoopsDelayRange	0	SFFloat				
triggerDistance	-1	SFFloat				
primeDistance	-1	SFFloat				

VRML Quick Reference (continued)

VRML Ver. #	Node	Field	Default Value	Value Type	Value Range	Traveler Implemented
1.0 + DS	Dspace_Spin	<i>Specifies a constant 3D spinning about an arbitrary axis through the origin.</i>				Full Support
		rotation		SFRotation		
1.0 + DS	Dspace_Sprite	<i>Represents a 2D bitmap that always faces the viewer.</i>				Full Support
		url	""	SFString		
		colorCount	128	SFLong		
		width	1	SFFloat		
height	1	SFFloat				
1.0 + DS	Dspace_UserPortal	<i>Sets subsequent WWW Anchor to be a user portal</i>				Full Support
1.1d	Background	<i>Specifies color backdrops that simulate ground and sky and a tiled texture.</i>				Partial Support
		groundColors	[]	MFCOLOR	First value only.	
		skyColors	[0 0 0]	MFCOLOR	First value only.	
		scenery	""	MFString	First value only.	
1.1d	Environment	<i>Defines global environmental attributes, such as ambient lighting.</i>				Partial Support
		ambientIntensity	0.2	SFFloat	standard	
		ambientColor	1 1 1	SFColor	standard	
		attenuation	0 0 1	SFVec3f	ignored	
		fogType	NONE	SFEnum	ignored	
		fogColor	1 1 1	SFColor	ignored	
fogVisibility	0	SFFloat	ignored			
1.1d	WorldInfo	<i>Contains information about the world</i>				Full Support
		title	""	SFString		
		info	"<Undefined info>"	MFString		

C.2 DSS to VRML Quick Reference

The following table is a quick reference mapping of the components used when coding in DSS or VRML. The bolded code in each reference represents the component used in DSS or VRML.

Component Name	DSS Syntax	VRML Mapping
<i>Space/Scene</i>		
General Properties (Required)		
Script type	< DSS ,1.0,"access_server","title",max_avatars,max_instances>	# VRML V1.0 ascii
Version	< DSS ,1.0,"access_server","title",max_avatars,max_instances>	# VRML V1.0 ascii
Access server	< DSS ,1.0," access_server ", "title",max_avatars,max_instances>	server field in DSpace_Server node
Title	< DSS ,1.0,"access_server", " title ",max_avatars,max_instances>	title field in WorldInfo node
Maximum number of avatars	< DSS ,1.0,"access_server", "title", max_avatars ,max_instances>	DSpace_MaxAvatars node
Maximum number of instances	< DSS ,1.0,"access_server", "title",max_avatars, max_instances >	DSpace_MaxInstances node
<i>Paths</i>		
Main script path	HOME_ADDRESS : "url"	url field in DSpace_Server node
Texture maps path	BMP_PATH : "url"	included in filename field in Texture2 node
Environmental Properties		
Bounding box	BOUNDING_BOX : minX, minY, minZ, maxX, maxY, maxZ	min and max fields in DSpace_CubeBoundary node
Collision	COLLISION : ON OFF	Set by Traveler for portals and links
Field of view	FIELD_OF_VIEW : degrees	Set by Traveler to 90 degrees
Ambient light	AMBIENT_LIGHT : red, green, blue	ambient Intensity and ambientColor fields in Environment node
Type	TYPE : FULL HALF	Implied by fields in Background node
Background colors	BACKGROUND_COLORS : number	Set by Traveler to 44
Sky color	SKY_COLOR : red, green, blue	skyColors field in Background node
Ground color	GROUND_COLOR : red, green, blue	groundColors field in Background node
Sky tile	SKY_TILE : "url"	scenery field in Background node

DSS to VRML Quick Reference (continued)

Component Name	DSS Syntax	VRML Mapping
Horizon maps	HORIZON_ELEMENT: "url", angular_width, bearing, elevation	Dspace_Horizon node
Other keywords		
Macro	INCLUDE: "url"	name field in WWWInline node
End of file	END:	-
Objects in Space/Scene		
Directional Lights	DIRECTIONAL_LIGHT: red, green, blue, vectorX, vectorY, vectorZ	Directional Light node
Properties		
Intensity	DIRECTIONAL_LIGHT: red, green, blue, vectorX, vectorY, vectorZ	intensity and color fields in DirectionalLight node
Direction	DIRECTIONAL_LIGHT: red, green, blue, vectorX, vectorY, vectorZ	direction field in DirectionalLight node
3D Geometry	PROP: "object_name", "url"	Shape nodes
Properties		
Name	PROP: "object_name", "url"	DEF/USE (not required)
URL/Data	PROP: "object_name", "url"	name field in WWWInline node or a shape node
Shading quality	PROP: "object_name", GOUBAUD FLAT UNLI T_FLAT ...	diffuseColor and emissiveColor fields in Material node and creaseAngle field in ShapeHints node
Color	COLOR_PROP: "object_name", red, green, blue	diffuseColor and emissiveColor fields in Material node
Texture maps	embedded in the prop (3DS file)	Texture2 node
Local sound	PROP_SOUND: "object_name", "url", volume,attenuation_circ umference	Dspace_Sound node
Collision	COLLIDES: "object_name", "ON" "OFF"	Set by Traveler for portals and links
Portal/Link	PORTAL_DESTINATION: "object name", "url"	WWWAnchor node
User portal	USER_PORTAL: "object_name", "url"	Dspace_UserPortal node

DSS to VRML Quick Reference (continued)

Component Name	DSS Syntax	VRML Mapping
Move/Position	MOVE_PROP: "object_name", deltaX, deltaY, deltaZ	Transform or Translation nodes
Rotate	ROTATE_PROP: "object_name", degreeX, degreeY, degreeZ	Transform or Rotation nodes
Scale	SCALE_PROP: "object_name", scaleX, scaleY, scaleZ	Transform or Scale nodes
<i>Animation</i>		
Spin	SPIN_PROP: "object_name", spinX, spinY, spinZ	Dspace_Spin nodes
2D Sprites	SPRITE_PROP: "object_name", "url", num_colors, width,height	Dspace_Sprite node
<i>Properties</i>		
Name	SPRITE_PROP: "object_name", "url", num_colors, width,height	DEF/USE (not required)
URL/Data	SPRITE_PROP: "object_name", "url", num_colors, width,height	url field in Dspace_Sprite node
Number of colors	SPRITE_PROP: "object_name", "url", num_colors , width,height	colorCount field in Dspace_Sprite node
Width	SPRITE_PROP: "object_name", "url", num_colors, width ,height	width field in Dspace_Sprite node
Height	SPRITE_PROP: "object_name", "url", num_colors, width, height	height field in Dspace_Sprite node
Local sound	PROP_SOUND: "object_name", "url", volume, attenuation_circumference	Dspace_Sound node
Collision	COLLIDES: "object_name", "ON" "OFF"	Set by Traveler for portals and links
Portal/Link	PORTAL_DESTINATION: "object_name", "url"	WWWAnchor node
User portal	USER_PORTAL: "object_name", "url"	Dspace_UserPortal node

DSS to VRML Quick Reference (continued)

Component Name	DSS Syntax	VRML Mapping
<i>Transforms</i>		
Move/Position	MOVE_PROP : "object_name", deltaX, deltaY, deltaZ	Transform or Translation nodes
Ambient Audio	Ambient Audio : "url", innerLoopMinCount, innerLoopCountRange, innerLoopMinDelay, innerLoopDelayRange, outerLoopMinCount, outerLoopCountRange, outerLoopMinDelay, outerLoopDelayRange, leftVolume, rightVolume, channelVolume	DSpace_Sound or DSpace_AmbientAudio nodes