ThreeSpace
REQUIREMENTS DOCUMENT

Adam Chodorowski & John Nilsson

June 28, 2001

Contents

Chapter 1

Introduction

1.1 Purpose

The purpose of this software requirements document is to provide a specification of what ThreeSpace shall and shall not do. The intended audience of this document is the software developers that will design and implement this system.

1.2 Scope

ThreeSpace is a 3D drawing program. It is a polygon object modelling and expression visualizer program. With TreeSpace it will be possible to create and modify 3D shapes using a score of different tools and visualize them in a 3D environment. In addition to general shapes ThreeSpace shall be able create objects based on mathematical expressions, such as $f(x, y) = sin(x) * cos(y)$. These mathematical expressions may also be applied to already created objects as filters to modify their appearance in some way. This can include structure, coloring, location, and/or orientation.

ThreeSpace is mainly intended to be used as a tool for visualizing complicated 3D functions. The primary audience is therefore freshmen collage students studying calculus, but the program shall not be so complicated as to limit the audience to these students.

1.3 Abbreviations and Definitions

1.3.1 Abbreviations

- **2D** - 2 dimensions.
- **3D** - 3 dimensions.

1.3.2 Definitions

This document and ThreeSpace uses the the following general 3D terms:

- **Dimension** - In ThreeSpace we are mainly concerned with two and three dimensions, since this is what the human eye and mind can percieve. In 2D we have two axis of orientation (eg. width and height) and in 3D we have three axis of orientation (eg. width, height and depth).
- **Coordinate system** - In ThreeSpace we are concerned with mainly two coordinate system, namely cartesian and spherical. Cartesian coordinates in 3D are defined as a set of three real numbers x,y,z. Each set x,y,z is uniquely defined. Spherical coordinates in 3D are defined as

a set of two angles and a length. In this documentation and in ThreeSpace itself coordinates are always given in the cartesian coordinate system, unless otherwise noted.

- **Point** - A point is a coordinate in space represented by a tuple of numbers which specifies how it is located in respect to the origin. Since ThreeSpace is a 3D program, all points are defined in 3D, and hence 3-tuples are used to represent them. The first component in the tuple is refered to as x, the second as y and the third as z. Note that this is specified in terms of the cartesian coordinate system; and can easily be converted to spherical coordinates which use two angles and the distance from the origin instead.

- **Origin** - The reference point to which all coordinates are given in relation to. In the cartesian coordinate system this point is refered to as (0,0,0).

- **Vector** - A 3-tuple of numbers representing direction and length, without any indication of location.

- **Normal** - The normal of a plane or line is a vector defined to have the same direction as a line that is at a straight angle to the plane or line.

- **Polygon** - A polygon is a planar area in 3D space consisting of of three *vertices*, three *segments* and a *surface*. The *vertices* are points in 3D space which define the shape of the polygon, such that the *surface* of the polygon has the same normal as the plane that intersects all three *vertices* and is constrained to only be visible inside the *segments* of the polygon. The *segments* are defined as lines between the *vertices* (only two *segments* may have endpoints in any given *vertex*).

- **Object** - An object is a collection of polygons that is handled as a single entity. An object has a shape in 3D. The location of an object is given by a single point and all the polygons of this object are specified relative to this point.

- **Shape** - The shape of an object is the way an object looks. The shape of an object is specified by its polygons and their orientation.

- **Primitive object** - An object which has a general shape, such as a box, sphere, pyramid, prism, cylinder or such. Often objects with complex shapes can be constructed out of primitive objects.

Definitions pertaining to the visualisation of the 3D space:

- **Scene** - The scene is the 3D space in which all objects are created. It encompasses all currently visible and not visible objects that have been added to the 3D space by the user.

- **Render** - Since the computer screen are two dimensional, it is not possible to directly show 3D shapes with it. Therefore the scene must first be rendered, ie. the 3D shapes must be projected onto a 2D surface (namely the screen) in such a way that a illusion of depth is percieved. This is the process which converts the abstract data on the 3D object's shapes, color and other visual attributes into a viewable representation.

- **View** - A view is a rendered visualization of a specific portion of the scene. A view can be thought as being the picture taken by an imagenary camera positioned within the scene. The camera has an orientation that specifies how it is tilting (ie, which way is "up") defined by the *sky vector*. The camera is oriented in a specific direction that whic is specified by the *look vector* and has a *position* given as a coordinate in 3D which defines the position from which the view is rendered. Thus, if you want the camera to show a view of what's behind it you change the *viewing direction* and if you want to look at something closer up you move the *viewing position* closer to the object and if the object is upside down and you can change the *sky vector*.

- **Viewing plane** - A plane normal to the camera's *look vector* whose imaginary x-axis or y-axis is parallell to the camera's *sky vector*.
- **Wireframe rendering** - A view in which the surfaces of the objects are not visible. Instead, only the *vertices* and the *segments* are drawn. This is often faster than a view rendered with surfaces visible.
- **Solid rendering** - A view in which the surfaces of the objects are visible.

Definitions specifically concerning ThreeSpace:

- **Camera state** - A program state in which the camera movement features are available.
- **Object state** - A program state in which the user-interface is adapted for the movement and modification of objects.

1.4 References

1. IEEE Std 830-1998 "IEEE Recommended Practice for Software Requirements Specification"
2. Java 2 SDK API Documentation, Sun Microsystems
   `http://java.sun.com/j2se/1.3/docs/api/`
3. Java3D API Documentation, Sun Microsystems
   `http://java.sun.com/products/java-media/3D/releases.html`
4. Java3D API Tutorial, Sun Microsystems
   `http://java.sun.com/products/java-media/3D/collateral/`
5. "Debugging Java", Will David Mitchell

1.5 Overview

The general layout of this document follows what is recomended in IEEE 830, and should therefore be familiar to people who have read other requirements documents. The basic structure of this document is split into two parts; the *overall description* and the *specific requirements*. The *overall description* describes the general factors that affect ThreeSpace and its requirements. It contains background information but not any state specific requirements. The *specific requirements* describe all software requirements in sufficient detail for software engineers to design and implement the system and for testers to verify that the final product satisfies them.

Chapter 2

Overall Description

2.1 Product Functions

The program shall be able to:

- Create and modify shapes in 3D.
- Create shapes based on mathematical functions, such as $f(x, y) = sin(x) * cos(y)$.
- Apply filters to objects.
- Move around in the scene and render the view based on camera position.
- Load and save shapes and the entire scene.
- Apply visual attributes, such as color, to objects.

2.2 Product Perspective

ThreeSpace should be self-contained and not rely on other products for functionality, with the exception of the Java runtime system.

2.2.1 System Interfaces

ThreeSpace shall use the Java 2 SE Runtime Enviroment and the Java 3D extension API.

2.2.2 User Interfaces

The program shall communicate with the user with the help of a graphical user interface (GUI) based on SWING components and a rendered 3D view. The GUI will consist of a main window and dialogs. The main window will have a *menu bar* which gives access to some program functionality, and a *toolbar panel* with buttons for the most important functions. It will also have a *view panel* which shows the 3D graphics.

At any given time, the program is in a specific state. Depending on the state, different functionality can be utilised by the user through the *property panel*. The panel will contain components, such as sliders or buttons, to control specific functionality in the given state.

The following program states shall be available:

- **Camera state** - Camera movement.
- **Object state** - Object selection, movement, rotation and attribute modification using filter or manually.

ThreeSpace shall be able to render in four different modes:

1. Point rendering, where only the *vertices* are shown.
2. Wireframe rendering, where only the *segments* are shown.
3. Solid rendering without any shading.
4. Solid rendering with gourad shading.

The user shall be able to select rendering mode when in *camera state* or *object state*.

Camera state

In *Camera state* it shall be possible to move around in the scene through input devices, either the keyboard or the mouse (or a combination of both). It shall be possible through movement to look at any object from any direction and at any angle.

Object state

In *Object state* it shall be possible to select several objects and group them into logical blocks, which can then be worked upon as if it was a single object. It shall be possible to ungroup objects that have earlier been grouped together. It shall be possible to group together already existing groups, thus creating a tree-structure of groups containing other groups.

It shall be possible to create shapes by selecting a shape from the menu.

Error messages

The program should prevent input errors by enforcing strict rules to input controls. In case of an error the program shall inform the user through a dialog. The dialog shall be provide both information to regular users and also debug-information for software developers. The program shall always use the same dialog to ensure that the user knows an error has occured. If the error is due to bad input by the user the information inputed by the user shall remain after the error message has been presented. The program should as far as possible try to recover from system errors and make it possible for users to save his/her work before the program exits.

2.2.3 Hardware Interfaces

No specific hardware shall be used directly by ThreeSpace. It shall however require a mouse, keyboard, display system and storage medium (or equivalent devices) that can be accessed through the Java Runtime Enviroments's interfaces.

2.2.4 Software Interfaces

ThreeSpace shall use the following software interfaces:

- Java 2 Standard Edition, version 1.3
- Java3D, version 1.2.1

2.2.5 Memory Constraints

ThreeSpace shall run on a computer with at least 10 MB available disk space and with 128 MB available RAM.

2.2.6 Operations

ThreeSpace shall be a mainly user-driven application in that it shall not support long periods of unattended, automatic, operation. All actions taken by the program shall be in direct response to the actions of the user.

ThreeSpace shall not provide any functionality for making and managing backups of data files, this is solely the user's responsibility. Neither shall ThreeSpace provide any functionality to recover data files that have been damaged.

2.3 User Characteristics

The intended user of ThreeSpace has the following attributes:

- Have a High School education or equivalent.
- Have atlest one year of experience with computers and/or 3D programs.
- Have good knowledge of first year collage calculus or equivalent.
- Have good knowledge of the English language.

2.4 Constraints

The program shall be designed and implemented in such a way that it shall be possible to move around in a moderately complex scene (ie. containing a maximum of 1000 polygons) in real time (ie. with at least 10 frames per second) on a SUN MicroSystems Ultra 1 workstation or equivalent.

The program shall be programmed in Java 2 SE, version 1.3 using the Java 3D API. It shall not use any operating system specific calls or side-effects for portability reasons.

ThreeSpace should try to maintain data integrity and should not have any memory leakage. In the event of a fatal error the program should try to inform the user and recover sufficiently to allow the user to save his/her work. Any fatal errors that occur while saving data will corrupt data and the program shall not be required to have any methods for handling this. However, if the Java Runtime Enviroment reports a fatal error the program should try to report this to the user, if possible.

2.5 Assumptions and Dependencies

Changes in the functionality or interfaces of the Java 3D API and/or the Java Runtime Enviroment may force changes in the requirements due the close the interaction between ThreeSpace and those programming enviroments.

2.6 Apportioning of Requirements

Filtering capabilities are built using plugins to allow third-party programmers to extend the system in the future. Therefore some filtering methods will be delayed to future versions.

Chapter 3

Specific Requirements

3.1 External Interface Requirements

3.1.1 User Interfaces

The user interface can be subdivided into five parts. All parts work as interfaces between the user and the program but in different ways. The five parts are:

- Menu bar
- Toolbar panel
- Property panel
- View panel
- Error dialog

Menu bar

The *menu bar* is a standard menu interface. The following menustructure should exist. Note that the entries specified in *italic* are dynamic and represent additions that can be done by the system at startup-time.

1. Project
   (a) New
   (b) Open...
   (c) Save
   (d) Save as...
   (e) Export...
   (f) About...
   (g) Quit
2. Object
   (a) New
      i. Circle...
      ii. Plane...
      iii. Sphere...
      iv. Hemisphere...

v. Function...
vi. *Creator plugin 6...*
vii. *Creator plugin 7...*
viii. *...*
ix. *Creator plugin n...*

(b) Load...

(c) Save as...

(d) Delete

(e) Hide

(f) Show
i. *A list of previously hidden objects by name*

(g) Color
i. *Color filter plugin 1...*
ii. *Color filter plugin 2...*
iii. *...*
iv. *Color filter plugin n...*

(h) Position
i. Snap to plane...
ii. Snap to grid...
iii. Lock position...
iv. *Position filter plugin 4...*
v. *Position filter plugin 5...*
vi. *...*
vii. *Position filter plugin n...*

(i) Rotation
i. Lock rotation...
ii. *Rotation filter plugin 2...*
iii. *Rotation filter plugin 3...*
iv. *...*
v. *Rotation filter plugin n...*

(j) Structure
i. Scale...
ii. Mathematical function...
iii. Hide polygons...
iv. *Structure filter plugin 4...*
v. *Structure filter plugin 5...*
vi. *...*
vii. *Structure filter plugin n...*

3. View

(a) Set background color...

(b) Hide/Show axis

Toolbar panel

The following buttons shall be available:

- Open
- Save
- Save as
- Switch to *camera state*.
- Switch to *object state*.
- Switch to *rendering mode 1*.
- Switch to *rendering mode 2*.
- Switch to *rendering mode 3*.
- Switch to *rendering mode 4*.

Property panel

Depending on the current state ThreeSpace is in, different information and controls shall be available in the *property panel*.

In *camera state* it shall be possible for the user to change different properties that are connected to the camera:

- Camera position.
- Camera rotation around the X-,Y- and Z-axis.
- Adding, removing or activating bookmarks of camera positions.

In *object state* it shall be possible for the user to change the following properties:

- Object name (ie. labeling objects).
- Object rotation around X-,Y- and Z-axis.
- Object position.
- Object coloring.
- Apply filter to change object rotation, position or coloring attributes.
- Apply filter to change object structure.

View panel

The *view panel* provides a 2D representation of the 3D scene as defined earlier in this document. It shall be possible to directly modify the scene (ie. the view or objects, depending on the current state) using the *view panel* and the mouse. The goal is for these functions to be intuitive and easy to use. Real-time feedback shall be used to create a fast learning curve. All actions performed on the view panel using the mouse shall also be available as controls in the *property panel* but with greater precision. Interactions with the *view panel* shall be considered as a way to do *hands on* modifications to the view or objects.

Error dialog

All error messages caused by invalid input or erronous interaction with the program shall be presented through a common error dialog that always has the same general layout. In the case of exceptions, the error dialog shall provide access to a *simple* and an *advanced* version of the error message. The *advanced* error message should include a stack trace and other debugging information that might be of value when debugging the program.

3.2 Software Product Features

3.2.1 Rendering modes

Purpose

Due to the varying performance of 3D viewing on different machines the program shall supply the user with the option of different rendering modes. Some modes may also be used to allow seeing through objects. The actual rendering modes that are available have been defined earlier (see **??**, on page **??**.

Stimulus/Response Sequence

To change the rendering mode the user shall use the apropriate buttons on the toolbar panel.

Associated Function Requirements

The user shall be able to change the rendering mode at any time in the main window, regardless if the program is in *camera state* or *object state*.

3.2.2 Camera Handling

Purpose

The program shall have some means of moving around in the 3D scene. Certain analysis and drawing require that objects are viewed from different angles. The program shall be able to view an object from every angle and from every location possible in 3D.

Stimulus/Response Sequence

When in *camera state*, mouse movement with a specific mouse button pressed shall cause the view to change by modifying the *viewing direction*. Mouse movement with a different specific mouse button pressed shall change the position by effectively moving back and forth in response to mouse movement, creating a zoom-motion. If the user selects and activates a previously stored *view bookmark*, the camera shall move to that exact position, with that exact viewing vector and with that exact tilt. Camera position and *viewing direction* modification through the *property panel* will provide precision to positioning and viewing.

Associated Function Requirements

Exact movement through *property panel* shall constrain *viewing direction* between -180 and 180 degrees around the x-, y- and z-axis. Movement through the *view panel* shall wrap the angles defining the *viewing direction* making -181 to 179.

The specific domains are:

- **Viewing-angle around y-axis, left/right** - (-180,180)
- **Viewing-angle around x-axis, up/down** - (-180,180)

- **Viewing-angle around z-axis, tilt** - (-180,180)
- **Viewing-position (x,y,z)** - $((-\infty, -\infty, -\infty) - (\infty, \infty, \infty))$

In reality the *viewing position* is constrained, but this constraint is by Java3D.

3.2.3 Object Creation

Purpose

ThreeSpace shall have functionality to create new primitive objects and place them in the scene at the user's request. This is required for creating content in the scene, which would otherwise be quite boring.

Stimulus/Response Sequence

The user selects the desired primitive shape by using the program's *menu bar* that shall contain a list of available primitive shapes under the `Object -> New` submenu. Next, the program shall open a dialog containing controls that can be used by the user to modify the new objects initial parameters. The exact fields that are available depends on the type of primitive object selected, and shall contain reasonable default values. If the user confirms the dialog, a new object with the shape and parameters specified shall be created at the origin of the scene.

Associated Function Requirements

If the object-creation plugin tries to create an object which can't be created in the 3D scene for some reason (eg. lack of memory), an error message will be shown.

3.2.4 Object Movement

Purpose

It shall be possible to move objects to different locations in the current scene. This is required for the user to be able to rearange the composition of the scene (and thus it's appearance), and to move newly created objects to the desired place.

Stimulus/Response Sequence

First, the user shall switch to *object state* if that state isn't the current one. Second, the user moves the desired object by dragging and dropping it to the desired position, using the right mouse button. The movement shall be locket to the plane orthogonal to the camera's current *look vector*. Using the middle mouse button, the user shall be able to move the object closer or further away from the camera, in the direction of the *look vector*. If the user wishes to move an object in a different direction, he/she shall have to change the *viewing vector*. More exact means of specifying the position (ie. using numeric coordinates) shall be available though the property panel.

Associated Function Requirements

Though no defined constraints exist in Java3D as to the size of the scene, the scene is not unconstrained. The position of the objects specified through the property panel will only allow exakt positioning of objects in a limited space of the scene, this space will be defined around the origin since most positioning will be around that point. The position of an object shall be saved.

3.2.5 Object Rotation

Purpose

It may sometimes be desirable to rotate an object around its three axis. Rotation may simply be a way of arranging objects but is essential for ThreeSpace to become useful.

Stimulus/Response Sequence

First, the user shall switch to object state if that state isn't the current one. Second, the user rotates the desired object by clicking on it with the left mouse button and rotating around the x- and y- axis. If the users wishes to perform exakt rotations then he/she shall be able to do this through the property panel. Through the property panel rotation around all three axis will be possible. Objects shall always be aware of their rotation in respect to their rotation when they were created. Thus all rotation-specific values in the property panel are relative to the way the object looked when created.

Associated Function Requirements

Exact rotation shall be specified with 0.1-precision and in degrees. All messages and information pertaining rotation shall be in degrees. The rotation of an object shall be saved.

3.2.6 Object Coloring

Purpose

Colors are excellent as visual aids in identifying and distinguishing objects in a crowded scene. It looks nice too.

Stimulus/Response Sequence

Coloring will only be possible through the property panel or by applygin filters. Colors will be selected by their HSV- or RGB-values.

Associated Function Requirements

The domain of HSV-values and RGB-values are [0,255]. The color of an object shall be saved.

3.2.7 Filters

Purpose

ThreeSpace shall support the notion of filters that can modify the attributes of an object in more or less useful ways. The different types of filters that shall be available are the following:

- Position filters; changes the objects position in the scene.
- Rotation filters; changes the objects rotation angles.
- Structural filters; changes the polygon structure of the object.
- Coloring filters; changes the coloring of the object.

Filters are useful for automating tedious tasks and modifying objects in ways that would be almost impossible or very time-consuming by hand. It will actually be the most distinguishing feature of ThreeSpace. You will be able to do some seriously cool stuff with them.

Stimulus/Respone Sequence

To apply a filter to an object, the user first has to switch to the object state if not in that state already and then select the desired object. Depending on what type of filter the user wishes to apply, he/she shall activate the apropriate button on the property panel. For easy navigation, the filter button shall be grouped together with the other controls that modify the same sort of attributes. Ie., the button for color filters shall be placed close to the controls for manual color modification and so on. After activated the button, the filter dialog shall be opened. In this dialog there shall be a list of the currently applied filters to the current object (which are of the same type as the user selected). The dialog shall allow the user to add and remove specific filters. It shall also allow the user to modify filter-specific settings. To add a new filter, the user shall click on the addition button which will open a new dialog window in which the user can select what filter he wishes to add. To remove a filter, the user shall simply select the filter in the list and click on the removal button. Upon exiting the filter dialog the object shall immediately be redrawn if necessary.

Associated Function Requirements

The only limitation as to how many filters that can be applied to any single object shall be the amount of available memory and, ofcourse, the user's patience when it's time for rendering the objects.

3.2.8 Object Loading & Saving

Purpose

Sometimes it is desirable to save complex objects to be used in different scenes.

Stimulus/Response Sequence

The user will select an object in *object state* and then go to the menu bar and select Object -¿ Save as. A file dialog will appear for saving the object. Similarly, selecting Object -¿ Load will open a file dialog and the user may select the a ThreeSpace-object-file. The object will be appended to the current scene.

Associated Function Requirements

If, when saving an object, the file selected by the user already exists, the program shall prompt him/her for confirmation before overwriting it. If the file selected for loading is not a ThreeSpace-object-file then an error message will be shown.

3.2.9 Scene Saving & Loading

Purpose

It shall be possible to save the current scene to disk and later load it, possibly during a different session with the program. This is an essential feature, since the user would otherwise be forced to start over from scratch each time he/she starts a new session with the program. And that's simply not nice..

Stimulus/Response Sequence

The user will select Save/Load from either the toolbar panel or from the menu bar, a file-dialog will appear for the user to select a filename for loading/saving.

Associated Function Requirements

If the file specified in loading isn't a real ThreeSpace-scene-file then the load shall abort with an error message. The current scene shall not be removed. If loading a scene when a scene already exists, then the user shall be notified that the current scene will be replaced by the loaded scene. The save dialog should only appear if a filename has not been associated with the scene or if the user has selected save as. The program should auto-save the scene every 15 minutes.

3.2.10 Mathematical Object Creation

Purpose

ThreeSpace is intended, mainly, to be used as a viewing tool of mathematical functions. Therefore, it is essential that the program has native abilities to create these objects.

Stimulus/Response Sequence

The user selects Object-¿New-¿Function3D through the menu bar. A dialog appears with fields for entering the function and the function-domain. A new object is created based on this function and its function-domain.

Associated Function Requirements

The function must be of $f(x, y)$-style. The domain must be reasonable. The number of polygons of the object must not exceed polygon-constraints.

3.2.11 Scene Export

Purpose

Sometimes it may be desirable export the scene or the current view to a external format. Export as a JPEG image may be particularly useful if a certain view must be printed.

Stimulus/Response Sequence

The user selects the File-¿Export submenu and a list of export-plugins are shown. Each plugin shall provide a file-dialog for entering the name of the export-file.

Associated Function Requirements

If the file already exists, the user shall be notified.

3.3 Software System Attributes

3.3.1 Reliability

No guarantee of the software's reliability shall be given. However, the designers and implementers should strive for the highest reliability possible. The program shall be usable and not crash immediately upon starting.

3.3.2 Security

ThreeSpace shall not provide any specific security functionality such as cryptography or activity logging, since the intended application domain of this software is not a high risk one. It is up to the user to make sure that any classified information that has been entered into the program and later possibly saved is not accessed by unauthorized people.

ThreeSpace shall not access files that are outside the installation directory of the software without the explicit consent of the user.

ThreeSpace shall not use any kind of networking and/or automatically collect any data from the user's system other than is directly necessary for the software's operation.

3.3.3 Maintainability

ThreeSpace shall be designed and implemented in such a way as to increase it's maintainability. Major functionality of the software shall be designed as separate modules with clearly defined interfaces between them, so that future maintenance of the software can be done without resorting to complete rewrites. The filtering, loading and saving functionality shall be designed in such a way that new modules pertaining to those functions can be added later, perhaps by third-party developers, without modifying the program. In effect, a flexible plug-in system shall be designed and it's interface specification freely available.

3.3.4 Portability

ThreeSpace shall be portable to all systems that support the Java 2 SE Runtime Enviroment, version 1.3 or higher, and the Java3D extension API, version 1.2 or higher. The software shall not use any operating system specific calls or native interfaces since that will most likely break portability.