Three-Dimensional Printing on a Budget:
A Classroom-Friendly Technique for
Viewing Solid Objects

Design Specification

Diana Butter  Jeremy Garcia  Ryan Lewis
Tyler Nielsen

Sponsored by:
Michael Eisenberg
Department of Computer Science
University of Colorado, Boulder

9 December 2002
Project Proposal

This project will involve the creation of a software system for “affordable” 3D printing. The technique that we will use is based on a homespun classroom device first described in Cundy and Rollett’s 1951 book *Mathematical Models* [2]. Essentially, the device consists of a series of layers of transparency material, each of which sits on a flat shelf. When the transparencies are stacked so that each one depicts a horizontal slice of a 3D solid—and when the entire stack of transparencies is viewed under light from a distance—a remarkable view of the entire solid comes into view.

We will focus on the development of classroom-friendly software that students can use to print out sequences of transparencies that can be used in the Cundy-Rollett device. Students should be able to create a 3D surface or solid on the computer screen and then request a “print-out” of the solid in transparency layers that can then be used to view the structure.

Our goal in this project is to develop a free-of-charge software application that teachers and students can download from the Web, and that they can use to create, customize, and print out a wide variety of viewable 3D structures.
# Table of Contents

1 Introduction .............................................. 1
   1.1 Scope ............................................. 1
   1.2 Document Structure .................................. 1
   1.3 Overview of spectro ................................. 1
   1.4 Relevant Documents ................................ 4

2 User Interface ........................................... 4
   2.1 Slicer Menus ...................................... 4
       2.1.1 File Menu .................................. 4
       2.1.2 Edit Menu ................................ 5
       2.1.3 View Menu .................................. 6
       2.1.4 Help Menu .................................. 6
   2.2 Slicer Toolbars ..................................... 6
   2.3 3D View Toolbars ................................... 7
   2.4 Object Creation ..................................... 8
   2.5 Object Selection ................................... 8
   2.6 Object Manipulation ................................. 9
   2.7 Object Grouping ................................... 10

3 Design Overview ......................................... 10
   3.1 High-Level Modular Decomposition ................... 10
       3.1.1 win Package .................................. 10
       3.1.2 disp Package .................................. 11
       3.1.3 obj Package .................................. 11
       3.1.4 io Package .................................. 11
   3.2 Object Structure .................................... 11

4 spectro Classes and Interfaces ......................... 13
   4.1 win Package ....................................... 13
       4.1.1 MainWindow Class ............................. 13
       4.1.2 DisplayWindow Class .......................... 13
       4.1.3 ExtensionFilter Class ....................... 14
       4.1.4 MenuHandler Class ............................ 14
       4.1.5 StatusBar Class ............................... 15
   4.2 disp Package ....................................... 15
       4.2.1 Geometry Class ............................... 15
       4.2.2 Scene Class .................................. 16
       4.2.3 Render Class .................................. 17
   4.3 obj Package ........................................ 17
       4.3.1 ObjCreate Class ............................... 17
       4.3.2 TransCreate Class ............................. 18
   4.4 io Package .......................................... 19
       4.4.1 Parser Class ................................. 19
       4.4.2 ParserInterface Interface ................... 19
<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>File Descriptions</td>
<td>19</td>
</tr>
<tr>
<td>Summary</td>
<td>20</td>
</tr>
<tr>
<td>Glossary</td>
<td>21</td>
</tr>
<tr>
<td>References</td>
<td>22</td>
</tr>
</tbody>
</table>
1 Introduction

1.1 Scope

This document describes the design for \texttt{Spectre}. It describes the user interface, including menus and toolbars, as well as the modular decomposition of the system, the classes that compose \texttt{Spectre} and the format of the files it employs.

1.2 Document Structure

Section 1 introduces \texttt{Spectre} and describes the scope and structure of this document; Section 2 discusses the user interface of \texttt{Spectre}; Section 3 gives a high-level discussion of the modules composing the system; Section 4 gives a detailed description of the classes that make up \texttt{Spectre} including the methods provided by each class; and Section 5 describes the datafiles used by \texttt{Spectre}. A glossary of important terms and a list of related documents is provided at the end.

1.3 Overview of \texttt{Spectre}

This project will involve the creation of a software system for “affordable” 3D printing. The technique that we will use is based on a homespun classroom device first described in Cundy and Rollett’s 1951 book \textit{Mathematical Models} [2]. Essentially, the device consists of a series of layers of transparency material, each of which sits on a flat shelf; Figure 1 (from the original Cundy and Rollett book) depicts the series of shelves on which the transparencies will be placed. When
the transparencies are stacked so that each one depicts a horizontal slice of a 3D solid—and when the entire stack of transparencies is viewed under light from a distance—a remarkable view of the entire solid comes into view.

To take a specific example or two: if each layer of transparency were to depict a circle of constant radius, the user would see what appears to be a solid cylinder hovering within the device. If (on the other hand) the circles diminished in radius with increasing height, diminishing to a single point in the top layer, then the user would see a cone. These are simple examples, but there is no reason to believe that a wide range of useful three-dimensional structures could not be printed and viewed with this simple device. The effect of seeing these solids is really quite remarkable: it is rather like seeing a ghostly, shimmering object hovering in midair. The object may of course be viewed from all sorts of angles simply by walking about the device.

Why hasn’t this marvelous device been used more widely? When Cundy and Rollett first wrote their book, there were no transparencies (they used “Perspex” plastic), and actually creating the slides for a particular model was a time-consuming and complex process. As a result, their idea seems to have been forgotten for decades.

Our project will improve on the original idea: we will focus on the development of classroom-friendly software that students can use to print out sequences of transparencies that can be used in the Cundy-Rollett device. Students should (for instance) be able to create a 3D surface or solid on the computer screen and then request a “print-out” of the solid in transparency layers that can then be used to view the structure. There are many techniques that could, conceivably, be experimented with here: perhaps the device could be used for viewing crystal structures, quantum orbitals, topographical maps, “see-through” anatomical models, and so forth. It is possible that multicolor objects or lighting variations might make a certain degree of “animation” possible; given time, this could make an interesting experiment.

Our goal in this project is to develop a free-of-charge software application that teachers and students can download from the Web, and that they can use to create, customize, and print out a wide variety of viewable 3D structures.

A conceptual diagram of \textbf{spectre} is shown in Figure 2. First, the user designs a three-dimensional image with a GUI. This image can then be saved to a datafile and reused. Next, \textbf{spectre} takes the image and “slices” it at regular intervals to produce a set of pictures. These pictures are sent to the printer to produce the transparencies needed to produce a model of the original image.

Mike Eisenberg, the project sponsor, is a professor in the department of Computer Science at the University of Colorado, Boulder. His research interests include educational software, mathematics and science education, learnability of programming languages, and scientific computation. His work in educational software was the inspiration for \textbf{spectre}. He has worked on several projects intended for children; one of his most impressive creations is HyperGami, which allows children to design origami using a computer, then construct their inventions with paper.
Figure 2: Conceptual diagram of Spectre. The user designs three-dimensional images then generates a set of transparencies. The user can also save an image to a file for later processing.
1.4 Relevant Documents

The requirements specification for spectre is presented in the document spectre Requirements Specification.

2 User Interface

The User Interface of spectre is composed of two main windows: the Slicer window, and the 3D View window. The Slicer window displays a single “slice” of the current scene and performs the majority of tasks in spectre via a menu bar and tool bar. At the bottom of the Slicer window is a status bar that denotes the number of the current slice, the total number of slices, the current and total height (in inches), and the \( x \) and \( y \) coordinates of the mouse within the Slicer window. The 3D View window displays a 3D view of the current scene, and allows manipulation of objects and scenes via a simple tool bar. The view of the scene in the 3D View window is always at an isometric angle, replicating a view of the scene if printed out and placed in the physical viewing device. The current slice displayed in the Slicer window is denoted in the 3D View window via a transparent white rectangle. A screenshot of both windows is shown in Figure 3. It is incomplete as not all the features were implemented at the time of the screenshot.

The functions of the Slicer menus are discussed in Section 2.1, Slicer toolbar functions are discussed in Section 2.2 and 3D View toolbar functions are discussed in Section 2.3. Object creation, selection, manipulation and grouping operations are defined in Sections 2.4–2.7.

2.1 Slicer Menus

There are four menus: File, Edit, View and Help. The function of each menu item is described below. Keyboard shortcuts to each menu item are denoted in parentheses following their descriptions.

2.1.1 File Menu

The File Menu contains the following options:

**New** Discard the current scene and create a new one. (CTRL + N)

**Open** Open a previously saved scene. (CTRL + O)

**Close** Close the current scene.

**Save** Save the current scene to the currently active file. If no file exists yet, the function is identical to Save As... (CTRL + S)

**Save As...** Save the current scene to a file specified by the user.

\(^1\)In spectre, a scene is a collection of 3D objects.
Print Print the current scene as a set of transparencies, or save them to a PostScript file [1]. The user may select the number of slices to print and the number of copies to print. (CTRL + P)

Exit Leave spectre. (CTRL + X)

2.1.2 Edit Menu

The Edit Menu contains the following options:

Import Import an object from a saved scene file. The user selects the object to import by name from a list of objects within a chosen scene. Placement of imported objects is defined in Section 2.4. (CTRL + I)

Options Brings up a window allowing the user to edit specific options regarding the scene, as described below.

Scene Size Change the size of the scene by increasing or decreasing margins for printing as well as defining total height of the users device in inches. Changing margins will not cause cutoff; whatever is in the Slicer window will simply be scaled to print within margins defined. Also, changing the height of the scene will not change the number of slices within the scene. The height of the scene should correspond to the height of the viewing device.
Num. Slices Change the total number of slices in the scene. This will add to or subtract from the total number of slices to be viewed and printed in the scene while maintaining total defined height.

2.1.3 View Menu

The View Menu contains the following options:

View Slice Above Display the slice above the current slice in a sequence of slices. Slices are numbered from lowest to highest, so the bottom-most slice is slice number one. If the top slice is currently displayed, View Slice Above has no effect. (CTRL + U)

Top Display the uppermost slice in a sequence of slices. The number of the top slice is determined by the total number of slices, as defined in Options. (CTRL + T)

View Slice Below Display the slice below the current slice in a sequence of slices. If the bottom slice is currently displayed, View Slice Below has no effect. (CTRL + D)

Bottom Display the lowest slice in a sequence of slices. The number of the bottom slice is always 1. (CTRL + B)

Coloring Allows the user to choose between grayscale or RGB coloring for scene viewing and printing. The user cannot actually change the color of the object through this option.

2.1.4 Help Menu

The Help Menu contains the following options:

Tutorial Launch the tutorial program for spectre. This will guide the user through the basic usage of spectre and explain how to create and print a simple scene. (CTRL + H)

Manual Displays the complete user reference manual for spectre. (CTRL + M)

About Display a description of spectre, its authors, and current version.

2.2 Slicer Toolbars

The Slicer toolbar buttons provide a quick shortcut to the functionality contained within the menus in the Slicer window. They also provide the means of object creation, manipulation and grouping. Object manipulations and grouping are always performed on the currently selected object; object selection is defined in Section 2.5. Keyboard shortcuts are denoted in parentheses following each description.
Create A Scene  Function is identical to New in File Menu.

Open A Scene  Function is identical to Open in File Menu.

Save Current Scene  Function is identical to Save in File Menu.

Print Current Scene  Function is identical to Print in File Menu.

Bottom Slice  Function is identical to Bottom in View Menu.

View Slice Below  Function is identical to View Slice Below in View Menu.

View Slice Above  Function is identical to View Slice Above in View Menu.

Top Slice  Function is identical to Top in View Menu.

New Sphere  Creates a new sphere object in the current scene. Before the object is placed, user can define the size and color of the object. Placement of object is defined in Section 2.4. (ALT + P)

New Cone  Creates a new cone object in the current scene, with definitions of object identical to New Sphere. (ALT + N)

New Cylinder  Creates a new cylinder object in the current scene, with definitions of object identical to New Sphere. (ALT + L)

New Cube  Creates a new cube object in the current scene, with definitions of object identical to New Sphere. (ALT + B)

Import Object  Function is identical to Import in Edit Menu.

Translate  Allows user to move the currently selected object within the Slicer window. Object translation is defined in Section 2.6. (ALT + T)

Scale  Allows the user to increase or decrease the size of the currently selected object. Object scaling is defined in Section 2.6. (ALT + S)

Color  Allows the user to change the color of the currently selected object. Object coloring is defined in Section 2.6. (ALT + C)

2.3 3D View Toolbars

The 3D View toolbars provide a quick shortcut to some of the functionality defined in the Slicer toolbars, as well as some additional functionality not available in the Slicer. Keyboard shortcuts are denoted in parentheses following each description.

Scale  Functionality is identical to Scale in Slicer toolbar.

Color  Functionality is identical to Color in Slicer toolbar.
Translate X Allows the user to translate the currently selected object across the $x$-axis. Object translation is defined in Section 2.6. ($\text{SHIFT} + \text{X}$)

Translate Y Allows the user to translate the currently selected object across the $y$-axis. Object translation is defined in Section 2.6. ($\text{SHIFT} + \text{Y}$)

Translate Z Allows the user to translate the currently selected object across the $z$-axis. Object translation is defined in Section 2.6. ($\text{SHIFT} + \text{Z}$)

Rotate X Allows the user to rotate the currently selected object around the $x$-axis. Object rotation is defined in Section 2.6. ($\text{ALT} + \text{X}$)

Rotate Y Allows the user to rotate the currently selected object around the $y$-axis. Object rotation is defined in Section 2.6. ($\text{ALT} + \text{Y}$)

Rotate Z Allows the user to rotate the currently selected object around the $z$-axis. Object rotation is defined in Section 2.6. ($\text{ALT} + \text{Z}$)

Rotate Scene Allows the user to rotate the entire scene around the $z$-axis, allowing for a different view of the scene. Scene rotation is identical to object rotation, as defined in Section 2.6. ($\text{ALT} + \text{R}$)

2.4 Object Creation

Object Creation occurs when the user either clicks on one of the object creation buttons (New Sphere, New Cone, etc.) or when the user imports an object from another scene. In both cases, the coordinates of the object’s origin can either be defined numerically (in inches) or manually. If the user decides to define the coordinates manually, they can use either the keyboard or mouse. Using the mouse, the object’s $x$ and $y$ coordinates are defined by where the user clicks within the Slicer window, and the $z$ coordinate is defined by the current slice displayed in the Slicer window. Using the keyboard, the user moves a cursor with the directional arrows within the Slicer window, and changes slices using the “+” and “-” keys. The coordinates of the object’s origin are then set at the cursor and the object is created when the user hits the ENTER key.

2.5 Object Selection

Object selection is accomplished via the mouse or keyboard. Using the mouse, the user simply clicks on the object they wish to select. Multiple objects can be selected by holding the SHIFT key while clicking on objects. Using the keyboard, the user can cycle through the objects in the current scene by hitting the TAB key. Multiple object selection is not allowed with the keyboard. An object is denoted as selected in the Slicer window by a black and white outline and in the 3D View window by the object changing colors or “glowing”.
2.6 Object Manipulation

There are four ways to manipulate selected objects: rotation, scale, translation and color. When one of these manipulation types is selected, a dialogue box will appear containing only the selected object. The four types of manipulation are defined below.

**Rotation** Selected objects can be rotated around the \( x \)-, \( y \)-, and \( z \)-axes using the keyboard or mouse in the 3D View window, and can be rotated around the \( z \)-axis only in the Slicer window. In both cases, the user can either define a specific number of degrees to rotate, or they can rotate the object manually using the mouse or keyboard. The user can also use the keyboard to rotate by pressing the “<” and “>” keys to rotate the object in increments of 5 degrees. Rotation of grouped objects will be around an unspecified point guaranteed to be within the convex hull of the group of objects.

**Scaling** Selected objects can be scaled either numerically or manually. If the user decides to scale the object numerically, they input a percentage which will increase or decrease the overall size of the object. If the user decides to scale the object manually, they can use the “+” and “-” keys to increase or decrease the size of the object by 5%. They can also use the mouse to scale objects by clicking on the selected object and then moving the mouse away from the object’s origin to increase size and towards the object’s origin to decrease size, with the final manipulated size being defined by the size of the object when the mouse button is released.

**Translation** Translation of selected objects can be performed numerically or manually, within the Slicer or 3D View window. In the Slicer window, numerical translation is performed by inputting specific \( x \), \( y \) and \( z \) coordinates (in inches) to move the origin of the selected object to. Manual translation in the Slicer window is performed using either the mouse or keyboard. Mouse translation of \( x \) and \( y \) coordinates is accomplished by moving the selected object to where the user clicks within the Slicer window. Mouse translation of the \( z \) coordinate is not provided in the Slicer window. Keyboard translation is performed by using the directional arrows and “+” and “-” keys to move the origin of the selected object. In the 3D View window, the user can perform translations of objects within specific axes. Numerically they can define a specific value for a specific axis. Manually they can “slide” an object along the chosen axis using the mouse or keyboard. With the mouse, the object will move left or right along the \( x \) or \( y \) axes as the mouse moves left or right across the screen, as well as up or down along the \( z \) axis as the mouse moves up or down across the screen. The directional arrows on the keyboard will perform the same manipulations.

**Coloring** Changing the color of selected objects is defined via either inputting specific RGB values or selecting a color from a color wheel. If the current
coloring mode is set to grayscale, then the user can input a specific black value number or select a color from a grayscale bar.

2.7 Object Grouping

Multiple objects can be put into “groups” which are treated by \texttt{spectre} as single objects; any manipulations on the group is performed on all objects within the group. Selecting multiple objects and then using the \texttt{group} command will put them into a group. Multiple groups can be combined into a single group in the same way; this allows addition of single objects to predefined groups as well. When multiple groups are selected and grouped, a hierarchy is created, allowing the original groups to maintain their definition when ungrouped. Objects may be in only one group at a time. Selecting single or multiple objects and using the \texttt{ungroup} command will remove said objects from whatever super group they are in; if they were in a sub group they will be returned to that group.

3 Design Overview

This section gives an overview of the design of \texttt{spectre}. First it describes the high-level modular decomposition of \texttt{spectre}. Since \texttt{spectre} is very amenable to an object oriented approach, an overview of the object structure is given.

3.1 High-Level Modular Decomposition

Due to its object oriented nature, Java was selected as the language in which to implement \texttt{spectre} [3].\footnote{In addition to its object oriented capabilities, Java is also a logical choice because of its inherit cross-platform features.} The software will run on any platform implementing the Java 1.4.1 API [5]. The following design and interface descriptions will be written using Java syntax and terminology. Please consult the glossary for help with unfamiliar terms.

\texttt{spectre} can be divided into four packages: \texttt{win}, \texttt{disp}, \texttt{obj}, and \texttt{io}. Each package contains one or more related classes. These packages, and the tasks they perform, are described in the follow sections. The classes that constitute each package are described in later sections. Figure 4 shows the modular decomposition of \texttt{spectre}; refer to it throughout the following descriptions.

3.1.1 \texttt{win} Package

\texttt{win} is responsible for all window-related functionality. It creates the windows and responds to user input. \texttt{win} also contains classes to manage the menus and toolbars.
3.1.2 *disp* Package

The *disp* package manages a collection of three dimensional objects, called a scene, and displays them according to the user’s directions. To accomplish this, it defines classes which contain the objects, and query the operating system for display and printer properties that objects can use when they draw themselves.

3.1.3 *obj* Package

The *obj* package contains numerous factory methods for creating and manipulating objects. For example, methods such as `createSphere()` generate new 3D objects, while methods such as `createRotation()` generate transformations that can be applied to objects.

3.1.4 *io* Package

The *io* package contains a parser that reads a *spectre* file from disk to restore a previously-created scene. It can also be used to import objects from saved scenes.

3.2 Object Structure

This section has two purposes. The first is to explain the choice of an object oriented design for the implementation of *spectre*. The second is to give an overview of the design structure of *spectre* including which objects are instantiated during execution and how these objects interact.

An object oriented design was selected because this would allow *spectre* to have a very modular design. This has many benefits:
Figure 5: Object Instantiation

- It would allow for independence among modules. Each module can perform a task based on an API. This enforces that each module does not rely on the internal code of other modules.

- Since each module is independent from the others, it is easy to maintain. When a problem is identified, it can be easily resolved internally, leaving the API, along with other modules, unchanged.

- The polymorphic properties of object oriented design translate easily to the physical object representation that a scene must maintain.

To gain some insight into the way the objects in spectre are instantiated and work together, an overview of the object structure is presented here. Detailed descriptions of spectre’s classes and their methods can be found in Section 4.

Many objects are instantiated by spectre. Figure 5 is an acyclic directed graph showing the instantiation of each object and the subsequent objects created by them. Each node represents an instance of a particular class. A vector from one node to another node indicates that the first object instantiates the second. For example, the first node shows an instantiation of class MainWindow which then creates instances of class DisplayWindow.

As you can see, in Figure 5 there are three classes that are not instantiated by other classes. Parser and Geometry are both static classes. They are created at load time and any other class can then reference their static members. MainWindow is created when the program is started by the user. MainWindow
is instantiated at the beginning of the program, it is used this way to abstract execution as an applet or as an application.

On startup, MainWindow instantiates DisplayWindow. DisplayWindow is then responsible for all user display. To accomplish this, DisplayWindow creates Render, StatusBar, ExtensionFilter, and MenuHandler. These four classes are used to help retrieve and display information.

From this point, MenuHandler then handles the flow control, creating an instance of Scene when required. Finally, Scene is responsible for instantiating both ObjCreate, and TransCreate. This is the low level of our program where each object and transformation gets added to the scene.

4 **spectro** Classes and Interfaces

This section defines the public interface to each class. Classes are grouped according to their enclosing package.

4.1 **win** Package

4.1.1 MainWindow Class

The MainWindow class provides the startup capabilities for our program. It handles execution as both an applet and an application. It does this through the methods:

- **void main ( String [] args )**
  - Usage
    * First function to call when program is invoked as an application.
  - Parameters
    * args - Arguments passed in on command line.

- **void init ( )**
  - Usage
    * First function to call when program is invoked as an applet.

4.1.2 DisplayWindow Class

The DisplayWindow class provides an abstraction to the creation of user interface windows. This handles everything necessary to create and display windows to the user. This consists of a window to show a 2D slice (the Slicer), and a window to display a 3D image (the 3D Viewer). Its methods are:

- **void showWindow ( )**
  - Usage
    * Create and show all applicable user interface windows.
4.1.3 ExtensionFilter Class

The ExtensionFilter class extends the FileFilter Java class. This enables the user interface to limit the files displayed in Open and Save dialogs. Its members are:

- ExtensionFilter ( String ext, String description )
  - Usage
    * Constructor.
  - Parameters
    * ext - This will be used for matching files. An exact string match is performed to determine if there is a file match.
    * description - This string is displayed in the Open and Save dialogs to describe the filter.

- bool accept ( File file )
  - Usage
    * Determines if a specific file has the correct extension.
  - Parameters
    * file - File to check.

- String description ( )
  - Returns
    * Returns the description passed into the constructor.

4.1.4 MenuHandler Class

The MenuHandler class provides an abstraction to the menu bar and toolbar. It extends the AbstractAction Java class. This enables it to be passed into menu bar and toolbar constructors. Its members are:

- MenuHandler (DisplayWindow callback, String name, KeyStroke ks, String tooltip )
  - Usage
    * Constructor.
  - Parameters
    * callback - Associates an item with the window it modifies. This should be passed in as the window the MenuHandler was created in.
    * name - String to appear in the menu item associated with this action.
    * ks - A shortcut can be passed in, or null for no shortcut. KeyStroke is defined in the Java API.
* tooltip - When a mouse hovers over this item, this string will be
displayed to give the user more information about what action
the item will perform.

- void actionPerformed ( ActionEvent e )
  - Usage
    * This is used to respond to an action performed by the user.
  - Parameters
    * e - What event happened. ActionEvent is defined in the Java
      API.

4.1.5 StatusBar Class

The StatusBar displays scene and rendering information along the bottom of
the screen. Its members are:

- void setPane ( int number, String text )
  - Usage
    * Changes the text in a pane to the desired string.
  - Parameters
    * number - Which pane to modify.
    * text - What string to put in the pane.

4.2 disp Package

4.2.1 Geometry Class

The Geometry class provides an abstraction from inches to pixels. This manages
the conversions that all objects must do to provide an accurate scene. It contains
the following members:

- double inch2dist_x (double inches_x )
  - Parameters
    * inches_x - Distance in inches along the x-axis.
  - Returns
    * Returns the distance in pixels.
• double `inch2dist.y` (double `inches.y`)
  
  **Parameters**
  * `inches.y` - Distance in inches along the y-axis.

  **Returns**
  * Returns the distance in pixels.

• void `setTranslation` (Graphics2D `g`, Rectangle `r`)
  
  **Usage**
  * Set a new conversion to use.

  **Parameters**
  * `g` - Destination graphics image to map to. `Graphics2D` is defined in the Java API.
  * `r` - Source rectangle coordinates. `Rectangle` is defined in the Java API.

### 4.2.2 Scene Class

The `Scene` class handles the storage and retrieval of objects, as well as performing operations on lists of objects. It is implemented as a Singleton class. Its members are:

• Scene `getScene` ( )
  
  **Returns**
  * Returns the current scene, or null if no scene exists. Do not save this value, as it may change.

• void `initScene` ( )
  
  **Usage**
  * Clears the current scene, then creates a new empty scene.

• void `addBranchGroup` ( BranchGroup `branch` )
  
  **Usage**
  * Adds an object to the list of objects in the scene.

  **Parameters**
  * `branch` - What to add to the scene. `BranchGroup` is defined in the Java3D API [4].

• void `drawScene` ( Graphics `graph`, Rectangle `r`, Render `rv` )
  
  **Usage**
* Goes through the list of objects, drawing them on the Graphics object.

- **Parameters**
  * graph - Destination Graphics object.
  * r - Source dimensions for objects.
  * rv - Rendering options to use when drawing the scene.

- **Returns**
  * Returns a BranchGroup representing the entire scene. BranchGroup is defined in the Java3D API.

### 4.2.3 Render Class

The Render class is used for controlling what render options are set when drawing a scene. It has private members that can be accessed by the standard get/set functions. For example, a private variable “foo” has public members getfoo( ) and setfoo( value ). The private variables are:

- int curlayer - Layer currently displayed.
- boolean threeD - Draw in 3D.
- boolean grayscale - Draw image in grayscale.

### 4.3 obj Package

#### 4.3.1 ObjCreate Class

The ObjCreate class a static class that provides several methods for creating new objects. Each method takes arguments that describe the object to be created (such as height, width, or radius) and returns a data structure that represents the desired object. The methods are:

- Obj3D newEllipsoid ( double x_radius, double y_radius, double z_radius )
- Obj3D newCone ( double x_radius, double y_radius, double height )
- Obj3D newCube ( double x_len, double y_len, double z_len )
- Obj3D newCylinder ( double x_radius, double y_radius, double length )

All objects are initially centered about the origin, and in the case of cones and cylinders, oriented along the z-axis. Positions and orientations can be changed using transformations from the TransCreate class, discussed in the next section.

One additional method is provided to facilitate grouping of objects (or even groups of objects):

---

3The data structure will either be a Java 3D data structure, such as a BranchGroup, or a wrapper around a Java 3D data structure.
• **Obj3D group ( Obj3D [ ] objects )**
  
  **Usage**
  * Group an array of objects into a single object representing the entire group.
  
  **Parameters**
  * objects - Array of objects to be grouped.
  
  **Returns**
  * Returns a new object representing the entire group. Transformations applied to this group will affect all its constituent objects.

### 4.3.2 TransCreate Class

The **TransCreate** class is a static class that provides methods for creating transformations to apply to objects created using the methods of **ObjCreate**. The methods are:

- **Transform newTranslate ( double x_len, double y_len, double z_len )**
  
  **Parameters**
  * x_len - Distance to move the object along the x-axis, in inches.
  * y_len - Distance to move the object along the y-axis, in inches.
  * z_len - Distance to move the object along the z-axis, in inches.

- **Transform newScale ( double x_scale, double y_scale, double z_scale )**
  
  **Parameters**
  * x_scale - Scale factor to use along the x-axis.
  * y_scale - Scale factor to use along the y-axis.
  * z_scale - Scale factor to use along the z-axis.

- **Transform newRotate ( double x_rot, double y_rot, double z_rot )**
  
  **Parameters**
  * x_rot - Angle to rotate about the x-axis, in radians.
  * y_rot - Angle to rotate about the y-axis, in radians.
  * z_rot - Angle to rotate about the z-axis, in radians.

  The direction of rotation is determined by the right hand rule.\(^4\)

\(^4\)If your right thumb points in the direction of the positive rotational axis, then the fingers curl in the direction of positive rotation.
4.4  \textit{io} Package

4.4.1  \textbf{Parser} Class

The \textbf{Parser} class provides a single method:

- \textbf{void Parse ( string filename, ParserInterface pi )}

  \textbf{Parameters}
  \begin{itemize}
    \item \texttt{filename} - String representing the path to the file to read from.
    \item \texttt{pi} - A class implementing the \textbf{ParserInterface} interface, described below.
  \end{itemize}

4.4.2  \textbf{ParserInterface} Interface

A class implementing the \textbf{ParserInterface} interface must implement a method with the following prototype:

- \textbf{void add ( Obj3D object )}

  \textbf{Usage}
  \begin{itemize}
    \item This method will be called for every object parsed from a file by the \textbf{Parser} class. It should add the object to the current scene.
  \end{itemize}

5  File Descriptions

\texttt{Spectre} reads and writes ASCII data files. Each file defines a set of objects to create a scene. A file is a sequence of \textit{Object} definitions, where an \textit{Object} is defined as follows:

\begin{center}
\begin{tabular}{ll}
\textit{Object} & \texttt{object\_name \{} \texttt{Parameters} \texttt{\}} \texttt{;}
\textit{Parameters} & \texttt{Parameter \{} \texttt{Parameters} \texttt{\} ;}
\textit{Parameter} & \texttt{parameter\_name \texttt{=} parameter\_value}
\end{tabular}
\end{center}

For example, a simple data file describing a scene containing a Cylinder and a Sphere might look like this:

Cylinder: \{height=10, radius=4, center\_x=0, center\_y=1, rotation\_x=30, rotation\_y=0\};
Sphere: \{radius=4, center\_x=-4.5, center\_y=2.7,\};

Whitespace in the file is ignored. Note that this definition optionally allows a trailing comma, as shown in the Sphere definition above.

While this representation is sufficient for simple objects, more complex objects formed by extensive grouping and transformations can result in objects
that cannot be described in this manner. Such objects will be written in a format not easily read by humans. The definition of such an object will begin like the simple objects described earlier: \texttt{Obj3D: \{ object definition \}}. However, the structure of the object definition is still under development.

6 Summary

This document presented the design for \texttt{spectre}. It presented a high-level view of the user interface and modular decomposition, and provided a lower-level description of the external APIs of the classes that constitute \texttt{spectre}. Although the details will change somewhat during the implementation phase, the final product should strongly resemble the design expressed in this document.
Glossary

3D
Three Dimensional.

3D View
One of the two main windows in Spectre, the 3D View window allows the user to view a 3D projection of the current scene as it will appear in the physical viewing device.

applet
A Java program embedded in an HTML document, executed by loading the web page.

ASCII
The American Standard Code for Information Interchange.

ASCII File
A file written in ASCII.

constructor
A method called to create a new instance of a class.

GNU
A UNIX-compatible software system developed by the Free Software Foundation.

GPL
The Free Software Foundation’s general public license.

greyscale
A mapping of RGB colors onto a set of colors consisting only of black and white. In the RGB color scheme, such colors are represented as a triple where all three numbers are equal (i.e. \( (76, 76, 76) \)).

GUI
Graphical user interface.

HTML
The Hyper Text Markup Language. Used for creating web pages.

HyperGami
A computer-based origami creation kit.

package
In Java, a package is a collection of related classes.

PDF
The Portable Document Format used by Adobe applications.

PostScript
A language used for printing documents on laser printers.

RGB
Red, Green and Blue values. A set of three numbers (from 0 to 255) that define a specific color.

scene
A collection of 3D objects.

Singleton class
A class where the constructor is private, allowing the class to enforce that only one instance is created.

slice
A two dimensional cross-section of a three dimensional object.

Slicer
One of the two main windows in Spectre, the Slicer allows the user to view a single slice of the current scene.

Spectre
Simulated Photorealistic Educational Classroom Tri-dimensional Rendering Engine.

static class
In Java, a static class is a class whose members (both methods and fields) are all static. Only one copy of a static class is created. Such classes are useful for library functions and global constants.

transparency
A sheet of transparent plastic designed for use with overhead projectors.
References


