# A Graphics Pipeline for Directly Rendering 3D Scenes on Web Browsers

*Master's Thesis*


Edgar Marchiel Pinto

# A Graphics Pipeline for Directly Rendering 3D Scenes on Web Browsers

DISSERTATION

concerning to the investigation work done to obtain the degree of

MASTER OF SCIENCE

in

COMPUTER SCIENCE AND ENGINEERING

by

Edgar Marchiel Pinto
natural of Covilhã, Portugal

Computer Graphics and Multimedia Group
Department of Computer Science
University of Beira Interior
Covilhã, Portugal
www.di.ubi.pt

# A Graphics Pipeline for Directly Rendering 3D Scenes on Web Browsers

Author: Edgar Marchiel Pinto
Student Id: M1489

**Resumo**

Nesta dissertação propomos um pipeline gráfico, na forma de uma biblioteca Web-3D, para a renderização de cenas 3D directamente no *browser*. Esta biblioteca de código livre chama-se Glyper3D. Foi desenvolvida usando a linguagem de programação JavaScript, em conjunto com o elemento canvas do HTML5, permitindo a criação, manipulação e renderização de conteúdos 3D no *browser*, sem ser necessária a instalação de qualquer tipo de *plug-in* ou *add-on* para o *browser*, ou seja, não tira partido de aceleração gráfica. Esta é a principal diferença em relação a outras tecnologias Web3D. Como é uma biblioteca direccionada para um ambiente web, foi desenvolvida para proporcionar maior usabilidade, proporcionando assim uma forma mais simples e intuitiva para desenvolver conteúdos 3D directamente no *browser*. Glypher3D pode ser usada para melhorar uma página *web* em vários aspectos, pois permite a criação de logotipos em 3D, modelos geométricos, entre outros propósitos. É uma biblioteca multi-plataforma e funciona em todos os *browsers* compatíveis com o elemento canvas do HTML5, como o Firefox, Safari, Opera e Chrome.

Supervisor: Prof. Dr. Abel Gomes, DI, University of Beira Interior

# A Graphics Pipeline for Directly Rendering 3D Scenes on Web Browsers

Author: Edgar Marchiel Pinto
Student Id: M1489

**Abstract**

In this dissertation we propose a graphics pipeline, in the form of a Web3D graphics library, for directly rendering 3D scenes on web browsers. This open source Web3D graphics library is called Glypher3D. It is entirely written in JavaScript (together with the HTML5 canvas element) and aims at enabling the creation, manipulation and rendering of 3D contents within a browser, without the need of installing any type of web browser plug-ins or add-ons (i.e. it does not take advantage of hardware acceleration), which is the main difference when compared to other Web3D technologies. As a library intended for the web environment, it was developed having in mind usability, therefore it is a simple and more intuitive way to deploy 3D contents on browser. Glypher3D can be used to enhance an web page, by allowing the creation of 3D logos, models, advertisements, among other purposes. Its a multi-platform library and works in the HTML5 canvas-compatible browsers like Firefox, Safari, Opera and Chrome.

Supervisor: Prof. Dr. Abel Gomes, DI, University of Beira Interior

# Preface

First, I would like to thank to my supervisor Prof. Dr. Abel Gomes for all the patience, dedication and transmitted knowledge during the realization of this dissertation.

I am also thankful to all my family and friends for all the support provided during my academic and personal life.

Edgar Marchiel Pinto

# Contents

# List of Figures

# Chapter 1

# Introduction

This chapter describes what has led to this dissertation, by defining the concept of Web3D and explaining the impact and limitations of the actual Web3D technologies in the development of 3D contents for the web. Then, the research issues and contributions of the proposed Web3D library called Glypher3D, are briefly discussed. Finally, we show how this dissertation is organized.

## 1.1 Motivation

The 3D concept for the web has been around for some time, and it was created with the purpose of enhancing the web experience in many ways. With 3D on the web, the web experience becomes richer and more visually appealing, in areas that go from finance, to science, education, e-commerce and to entertainment (e.g. the 3D virtual world of Second Life), among others. For example, an online shop allows us to see and interact with the products in a realistic 3D way, which tends to maintain potential buyers more time on the website, and more interested in the product [26].

But, until recently, providing 3D contents in a web page was not an easy task. Limitations, such as bandwidth, hardware specifications or specific software, made it difficult to implement the 3D concept for the web. Nowadays, due to faster connections, better computers, and more advanced browsers, the Web3D concept is about becoming a reality [28].

The Web3D concept is used to describe the technologies, protocols, languages, and file formats supported by the World Wide Web (WWW) that allow us to develop 3D graphics applications for the web. Virtual Reality Markup Language (VRML) was the first language, and also the first International Organization for Standardization (ISO) standard developed for Web3D. This standard, although still in use, was later replaced by eXtensible 3D (X3D), an eXtensible Markup Language (XML)-based technology. However, other non standard technologies, like Java3D, Asynchronous JavaScript and XML (Ajax)3D, and the newest Flash3D, Canvas 3D JS Library (C3DL), 3D Markup Language for Web (3DMLW) and O3D, can also be classified as Web3D technologies [25].

However, some of these Web3D technologies, like VRML, X3D, Ajax3D, and Java3D, have failed to revolutionize the way of how the common web users surf the internet, and ultimately, the way of providing rich 3D web environments. That happened due to a variety of reasons. First, many users and developers are not aware of these 3D technologies, due to their limited use on the actual web. Second, they are not very popular and widely used technologies, due to the unattractive, time consuming and expensive way of creating 3D contents. Finally, they do not provide an easy and direct way (plug-in or add-on free) to create and interact with 3D environments within a browser.

The newer Web3D technologies (Flash3D, C3DL, 3DMLW and O3D), despite being innovative and providing different approaches for the use of 3D on web, or despite providing even more realistic 3D contents, also suffer from the big issue that is the installation of third party browser plug-ins or add-ons [26].

So, the biggest drawback of the mentioned technologies, is their dependence on browser plug-ins or add-ons to display and interact with 3D environments directly on browser. It is true that the plug-ins or add-ons have no cost and are easy to install, however, many of the web users, when faced with the obligation to install those plug-ins, lose their willingness to try the 3D experience or simply do not want to waste their time on that installation. It is also true that there are a lot of plug-ins available (especially for VRML and X3D). However, some of them do not work with the recent versions of the most common browsers, or they were developed to an specific browser, with a specific version, and for a specific operating system, with a specific version. For example, to deploy 3D contents on browser using the C3DL library, it is necessary to install an specific Firefox 3.5 add-on (Canvas 3D). This library only works with Firefox 3.5.

Therefore, because of this Web3D technologies dependence on third party browser plug-ins or add-ons, and to facilitate the integration of 3D contents directly on browser, this dissertation proposes an open source Web3D JavaScript library, called Glypher3D. This library was developed having in mind the web context, i.e. it is intended to be a simple and fast web application. It allows us to develop 3D contents within a browser, but without the obligation to install any kind of browser plug-ins or add-ons. For that, it uses the HyperText Markup Language (HTML) 5 canvas element drawing and rendering functions, together with 3D graphics algorithms encoded in JavaScript.

Notice that when this dissertation was proposed, the HTML5 canvas element was still evolving and was more limited than it is now. For example, the pixel-based manipulation methods, used by our Gouraud shading, were only supported by the Firefox 3 browser, and only recently were added to the newest releases of Safari, Opera and Chrome browsers. Also, Web3D technologies such as C3DL have evolved considerably in the last year, and O3D was introduced only recently in 2009.

## 1.2 Research Issues and Contributions

The open issues that have led to the creation of the Glypher3D library were the following:

- Is it possible to create an Web3D library that does not require the installation of any kind of browser plug-ins or add-ons, and that allows us to create and interact with 3D

contents directly on browser?

- By not using any plug-in or add-on, as those used by all the mentioned Web3D technologies to take advantage of software or hardware acceleration (through OpenGL[1] or DirectX [2]), how would it be the rendering performance of the library?
- Also, by not using any plug-in or add-on, would it be the HTML5 canvas element, together with JavaScript, the best technology to draw 2D graphics for the web?
- By using this technology, would Glypher3D be a cross platform and a browser-independent library?

So, the goal of this dissertation is to answer to those questions, by describing in what aspects Glypher3D differentiates itself from the most used Web3D technologies, as well as by explaining in detail the Glypher3D library architecture, rendering pipeline and all of its functionalities.

The main contribution of this dissertation is then to facilitate the integration of 3D contents within a web page, by proposing an open source Web3D graphics library (Glypher3D), that allows us to create, manipulate, and display 3D contents directly on browser, in a more intuitive and easier way (without browser plug-ins or add-ons) than other Web3D technologies.

## 1.3 Organization of the Thesis

This dissertation is organized into five chapters, as follows:

- Chapter 1. In this chapter we overview the Web3D concept and Web3D technologies. The research issues that led to the Glypher3D library, as well its contributions to the advance of knowledge, are briefly enumerated.
- Chapter 2. In this chapter, we explain in detail how the more relevant and used Web3D technologies work, and what results can be obtained with them.
- Chapter 3. In this chapter, we describe and explain in detail the HTML5 canvas element, the Glypher3D architecture, its rendering pipeline and all of its functionalities.
- Chapter 4. In this chapter, we show how to use Glypher3D in the development of Web3D applications. It is also tested the rendering performance of this library on different platforms and browsers.
- Chapter 5. In this last chapter, we draw some conclusions from the development of the Glypher3D library, and indicate some directions for future work, in particular how to improve its performance.

[1] OpenGL is the most widely used and supported 2D and 3D Application Programming Interface (API). It allows hardware acceleration and is an open and multi-platform graphics standard [21].

[2] DirectX is a set of API's developed by Microsoft. It allows for graphics or multimedia applications, running on the Windows platform, to take advantage of hardware acceleration [30].

# Chapter 2

# 3D Web Graphics: The State-of-the-Art

This chapter describes the more relevant Web3D technologies to design, display and interact with 3D web contents. It will be discussed how the concept of Web3D evolved, since the creation of VRML to the widely used Flash platform, and even to the newest O3D. We also show how to integrate such technologies within a web page, and the results that can be obtained with each one of them in the development of Web3D applications. Also, these technologies will be compared to each other, and finally with Glypher3D.

## 2.1 VRML

Created in 1994, by the VRML Consortium, this high level 3D content development language was responsible for introducing the concept of Web3D, and was the first ISO standard for the creation and visualization of 3D contents on the Internet [43]. With a 3D environment on the web, VRML would enhance the web browsing experience in many ways. This Web3D technology has evolved through a series of releases:

- VRML 1.0.
- VRML 2.0 or VRML97.
- X3D.

The VRML 1.0, officially released in 1995, was proposed to be a common language for the creation of 3D scenes distributed over the Internet. For that, it was created with the intent of being a cross platform, extensible, and bandwidth conservative language, i.e. allowing the 3D contents to be accessible through multiple operating systems, multiple web browsers and distributed over low-bandwidth connections. These goals made VRML the foundation language for Web3D contents for the masses, because they allowed any web developers or web 3D enthusiasts to develop 3D contents, even if they had slow Internet connections or slow computers [43].

With this VRML release, developers started to code 3D scenes, and tool vendors started to build VRML plug-ins, standalone VRML players, and tools to deliver these scenes over the Internet or on desktop applications. Therefore, it can be said that VRML 1.0 brought the platform-independent 3D concept for the web. However, this release was very limited because it only allowed to create non-realistic and static 3D scenes, and was not possible to interact with the 3D objects within that scene [43].

Due to these limitations, and in order to provide a more immersive, realistic and interactive 3D world, a second major version of VRML was released, the VRML 2.0 (defined later as VRML97). This release brought support for interactivity, sound, animation, and ultimately the ability to create more complex 3D worlds, e.g. worlds with light sources, fog, etc [43].

It was in 1997 that ISO recognized the VRML97 specification as an international standard, although the Web3D consortium (old VRML consortium) had considered the VRML 1.0 and also VRML 2.0 specifications obsoletes [43].

The description of an VRML 3D world, with simple or complex objects, is stored in plain American Standard Code for Information Interchange (ASCII) text files, with the `.wrl` extension (short for "world"). A VRML world consists of nodes that are arranged into a scene graph data structure, a hierarchy of groups and shapes arranged like a family tree. Scene graph parents manage groups of children like shapes, lights, sounds, etc [43, 12]. Listing 2.1 shows how to encode a simple box geometric primitive node [31].

Listing 2.1: VRML 2.0 box geometric primitive node.

```
1  #VRML V2.0 utf8
2
3  Background {
4    skyColor [ 1 1 1 ]
5  }
6  Viewpoint {
7    description "Book View"
8    orientation -0.747 -0.624 -0.231 1.05
9    position -1.81 3.12 2.59
10 }
11 Shape {
12   geometry Box {
13     size 1 1 1
14   }
15   appearance Appearance {
16     material Material {
17         diffuseColor 1 0 0
18     }
19   }
20 }
```

In order to display, interact, and navigate on the VRML 3D world described in the `.wrl` file, we must use a VRML web browser plug-in or a standalone player to interpret this file. The `.wrl` file is parsed by the installed VRML web browser plug-in (player), and the 3D contents are rendered into the browser window [31]. VRML players use 3D graphics engines like OpenGL or DirectX to take advantage of the 3D graphics card in the client computer.

Listing 2.2 shows how to embed a VRML file within a web page, usually `index.html`. Notice that the VRML `Box.wrl` file must be in the same directory as the `index.html` file.

Listing 2.2: VRML file included in a web page through the HTML tag EMBED .

```
1 <html>
2   <head>
3     <title>VRML 2.0 Box geometric primitive node</title>
4   </head>
5   <body>
6     <embed src="Box.wrl" width="300" height="250">
7   </body>
8 </html>
```

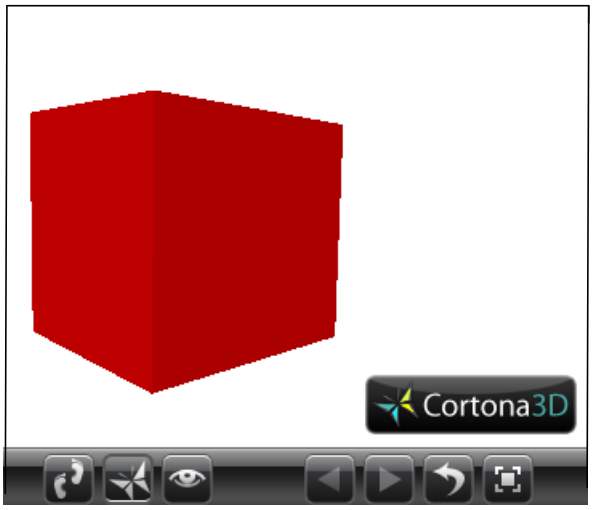


Figure 2.1: Box rendered through the VRML browser plug-in Cortona3D Viewer 6, with Firefox 3.0 on Windows XP SP3.

Figure 2.1 shows the 3D box stored in the `Box.wrl` file (see Listing 2.1). A more complex VRML 3D world is shown in Figure 2.2. This latter world includes lighting and shading.

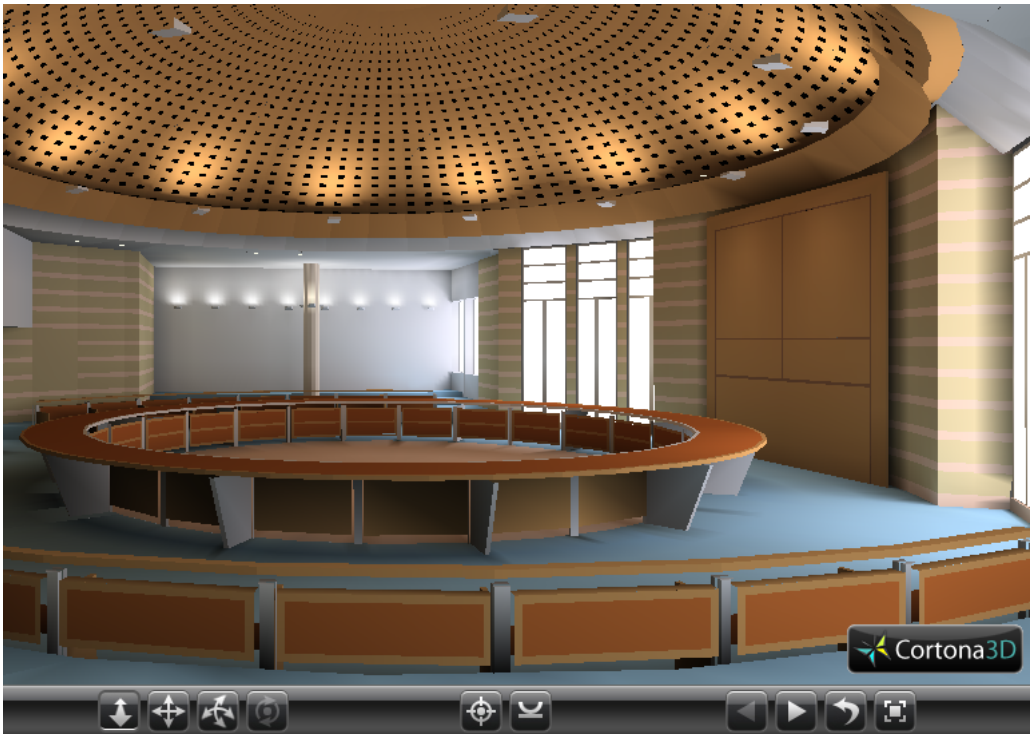


Figure 2.2: Virtual chamber rendered through the VRML browser plug-in Cortona3D Viewer 6, with Firefox 3.0 on Windows XP SP3. Retrieved from [32].

VRML allows the creation of full 3D environments, but it has some limitations. For example, it does not allow for video streaming, binary compression, and multi-texturing. However, VRML inspired and led to the development of newer Web3D technologies, including the standard X3D (formerly known as VRML Next Generation (VRML-NG)). X3D is built on top of VRML and is understood as the newest generation of this technology [43].

## 2.2 X3D

This technology, developed by the Web3D consortium as the third generation of VRML, became a ISO standard in 2004, and is since then the only open and royalty-free ISO standard for the development of rich, interactive, animated and realistic 3D contents over the web. It was developed with the main goal of ridding off the deficiencies of VRML, and to make the creation of 3D graphics an easier and more intuitive task, accessible to a wide range of developers, including 3D graphics programmers and even non-programmers. With X3D, developers should be able to create 3D environments without understanding the low-level graphics API's [43]. Other goals in the development of X3D were:

- Maintain the compatibility with the previous VRML technology.
- Be an platform-independent web oriented standard.

To fulfill these goals, X3D defined three equivalent encoding formats: a classic VRML version, an XML-based version, and a compressed binary version to encode X3D scene files.

The classic VRML encoding maintains the same structure as the one of VRML97. The XML encoding allows that the X3D contents can be used by a broader audience, devices and applications, on different platforms across the web. Finally, the compressed binary encoding makes the file size smaller, and increases the parsing speed, the transmission, and the loading times [10, 26].

With the XML encoding, X3D allows developers to create and manipulate 3D contents using only XML tags, as opposed to the classic VRML encoding or VRML97, which require a specific knowledge of the language to create 3D contents.

To encode information in an X3D scene, we can use the XML encoding file format `.x3d` or the classic VRML encoding file format `.x3dv`. Notice that the classic VRML encoding has the same syntax as VRML97, with only two differences: the classic VRML encoding supports X3D nodes, and the first line header changes from `VRML V2.0 utf 8` to `X3D V3.1 utf 8` (compare headers from Listings 2.1 and 2.3). The binary encoding file format `.x3db`, just compresses any X3D scene (`.x3d`, `.x3dv` or even `.wrl`) for faster scene loading at run time and network streaming, and adds (if desired) XML security for content protection. Listing 2.3, with the classic VRML encoding, and Listing 2.4, with the XML encoding, show the commands necessary to create the same box of Figure 2.1 [9, 10].

Embedding an X3D file within a web page is done just like in VRML (Listing 2.2).

Listing 2.3: Classic VRML (`.x3dv`) encoding.

```
1  #X3D V3.1 utf8
2  Background {
3    skyColor [ 1 1 1 ]
4  }
5  Viewpoint {
6    description "Book View"
7    orientation -0.747 -0.624 -0.231 1.05
8    position -1.81 3.12 2.59
9  }
10 Shape {
11   geometry Box {
12     size 1 1 1
13   }
14   appearance Appearance {
15     material Material {
16         diffuseColor 1 0 0
17     }
18   }
19 }
```

Listing 2.4: XML (.x3d) encoding.

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE X3D PUBLIC "ISO//Web3D//DTD X3D 3.1//EN"
"http://www.web3d.org/specifications/x3d-3.1.dtd">
<X3D profile='Interchange' version='3.1' xmlns:xsd=
'http://www.w3.org/2001/XMLSchema-instance'
xsd:noNamespaceSchemaLocation='http://www.web3d.org/
specifications/x3d-3.1.xsd'>
  <Scene>
    <Background skyColor='1 1 1'/>
    <Viewpoint description='Book View' orientation=
    '-0.747 -0.624 -0.231 1.05' position='-1.81 3.12 2.59'/>
    <Shape>
      <Box size='1 1 1'/>
      <Appearance>
        <Material diffuseColor='1 0 0'/>
      </Appearance>
    </Shape>
  </Scene>
</X3D>
```

X3D uses a tree-structured scene graph to represent the graphics nodes that make part of the 3D world. This scene graph includes the geometry, appearance, animation and event routing. In addition to the geometry and animation behaviors expressed in XML, X3D allows scripting (mainly through JavaScript programming) and node prototyping, that together provide support for scene graph extensions, like complex animations and user interactions. Other advanced X3D functionalities, that VRML does not support, are multi-texturing surfaces, Non-Uniform Rational Bézier Spline (NURBS) parametric surfaces, geospatial positioning, interchangeable Humanoid Animation (H-Anim) bodies, and the IEEE Distributed Interactive Simulation (DIS) network protocol [9, 10, 12].

Just like VRML, X3D needs a player to parse and render an encoded X3D scene, which may also allows for user interaction and object animation. In fact, every browser needs a X3D player plug-in in order to render X3D scenes. These players can also be delivered as standalone or desktop applications. For a full description of the available VRML and X3D players see Appendix B. Figure 2.3 shows the architecture of an X3D player.

An X3D player has parsers that read the X3D files formats encodings. After that, the nodes are created and sent to the scene graph manager, which is responsible to manage the geometry, appearance, location and orientation of objects. Then, the scene graph is rendered. On the other hand, the event graph is responsible of all animation nodes. These animations can be extended or enhanced through scripting languages like JavaScript. The Scene Authoring Interface (SAI) enables that an application external to the X3D player can perform operations inside the X3D scene during run time. HTML web pages or external applications can embed X3D players to display and interact with the 3D objects [9, 10].

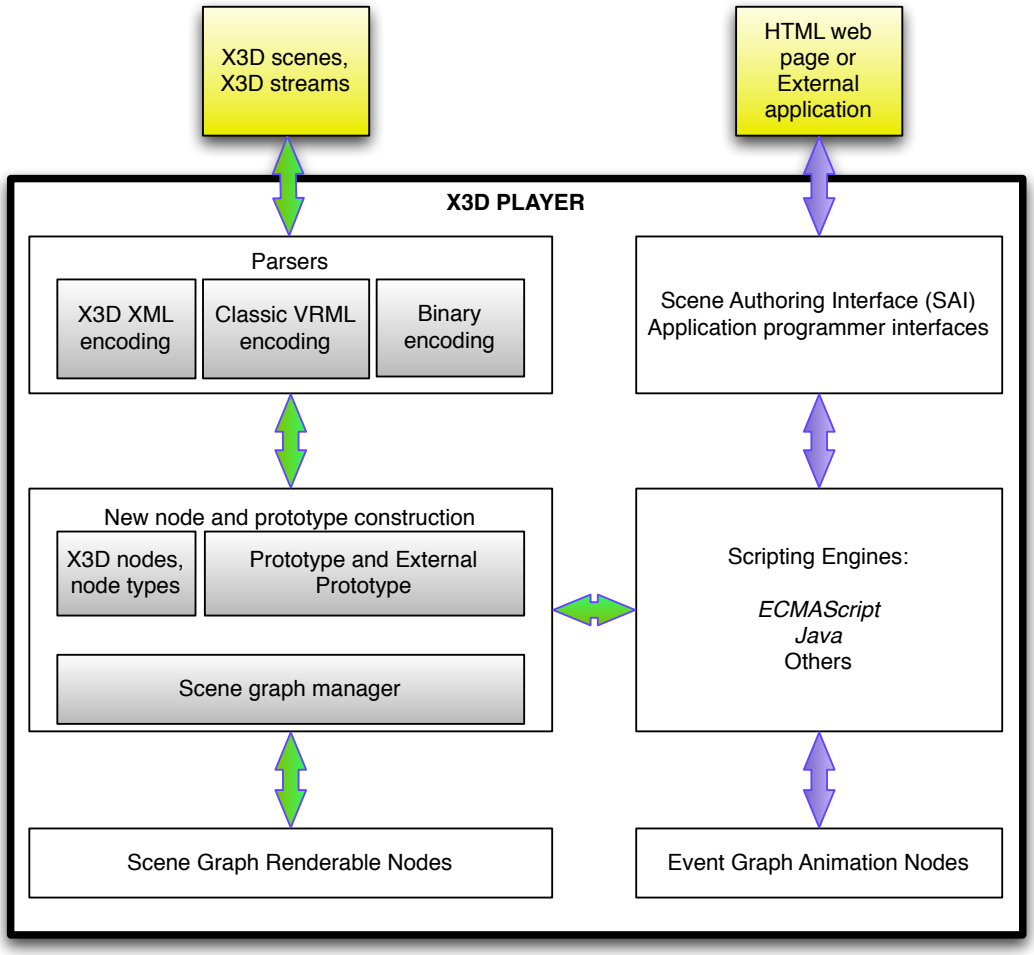


Figure 2.3: X3D player architecture. On the left, file parsing and rendering process. On the right, animation and scene graph manipulation. Image based on the model from the book [9]).

Most of 3D players use the low level OpenGL or DirectX rendering engines to render the graphical content. When hardware graphics acceleration is not available we use an software renderer instead. When choosing VRML and X3D players, developers must choose the one compatible with their operating system and browser version. Another aspect to take into account is whether or not a X3D player is proprietary or open source and royality-free. Even with the many resources, repositories, and material available to facilitate the construction of X3D worlds, this can become a complex task when creating complex and high detailed X3D scenes. So, to help creating X3D worlds, a variety of authoring tools can be used (see Appendix B). In Figure 2.4 is shown a high detailed X3D world.

The purpose of building X3D worlds, especially for the web, is to provide a better, more

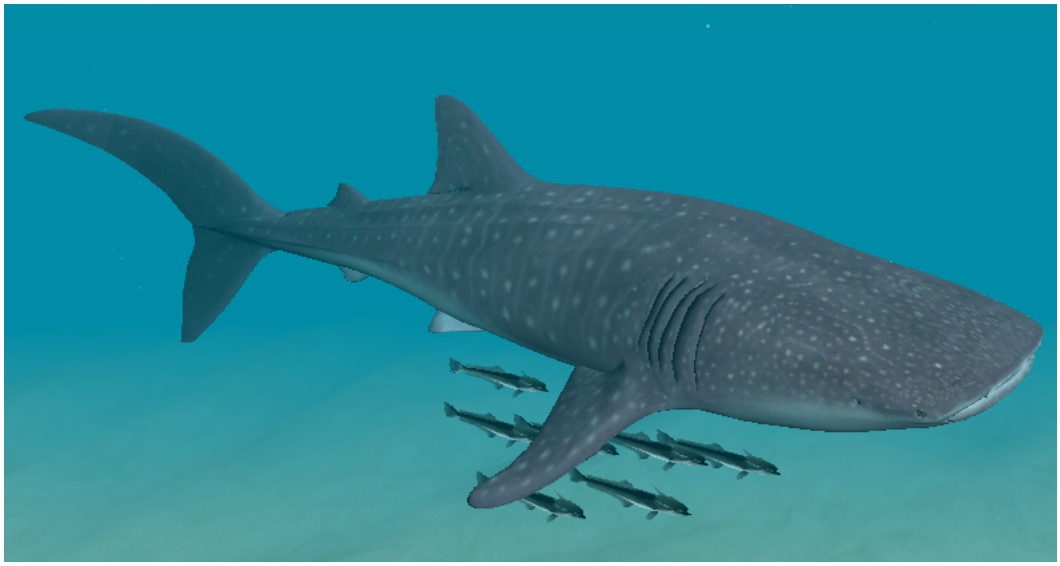

Figure 2.4: Shark rendered through the X3D browser plug-in Octaga Player 2.30.05, with Firefox 3.0 on Windows XP SP3. Retrieved from [32].

appealing and interactive web browsing experience. Improving this browsing experience makes people stay longer on websites and ultimately makes them to interact, experience or buy products showcased in an 3D virtual environment. X3D can be used in social networking, e-commerce, visual simulations, web pages animations, and in scientific, educational or medical applications.

X3D is the standard for the implementation of high detailed and interactive 3D environments across the Internet, and continues to grow due to the active support of the Web3D Consortium, as well as, due to the support of developers and users of Web3D technologies. However, these technologies are not yet widely used by web developers who want to deploy 3D contents in browser, mainly because that requires a solid knowledge of the VRML language or knowledge on how to use the X3D XML tags.

## 2.3 Java3D

Java3D, officially released in 1998, is a cross platform API that enables the development of 3D graphics applications using the popular Java programming language. It is considered the 3D extension for Java. It allows developers to create complex and interactive 3D desktop applications, or web based 3D applets, that can work efficiently on multiple platforms. With Java3D, developers can build up 3D scenes using either direct programming or loading 3D contents from external files, like VRML, 3ds Max (3DS), etc. Java3D provides a set of features, like building shapes, animation, user interaction, lighting, texturing, collision detection, sound, etc. Java3D applications take advantage of OpenGL or DirectX hardware graphic rendering engines [43].

As opposed to VRML, that is an human readable 3D content development language, stored in plain ASCII text files, that are interpreted by VRML players, and unlike X3D, that can use classic VRML, XML or a compressed binary version to encode X3D scenes, also interpreted by X3D players, Java3D is an extension to Java. It is a fully high-level 3D API. Java3D source code is only human readable while being written. The source code is then compiled into a platform-independent Java byte-code format that, as an intermediary form of code (not human readable and not binary machine code), provides the platform independence. This Java byte-code format must pass through a Java Virtual Machine (JVM), on the client side, to be converted into machine code in order to be executed. Because this conversion occurs at runtime, the Java byte-code can be executed in any platform, as long as a JVM is installed to allow the conversion and execution of the Java3D application. Because the Java byte-code format can be converted to machine code faster than the ASCII text of VRML files, in part due to its proximity to the machine code, Java3D applications end up to be executed faster than VRML applications [43].

Despite the mentioned differences, Java3D has some similar aspects with VRML and X3D. In fact, the Java3D API encapsulates the low level 3D details, allowing developers, even those without experience in 3D programming, to develop 3D applications. As said earlier, Java3D can load, among others, VRML files, allowing users to take advantage of the existing VRML models. Also like VRML and X3D, Java3D uses a tree-structured scene graph programming model, to store, organize and render all the components of an 3D scene. Java3D is also a royalty-free technology [43].

To use Java3D, their classes have to be downloaded separately from the standard Java distribution. This is to avoid a multiple set of options, that might be unnecessary for some users, in the standard Java distribution. After writing or downloading the source code of an Java3D application, a Java3D applet must be embedded within a web page, in order to run such application within that web page. This embedding process is illustrated in Listing 2.5. Notice that the class file `HelloJava3Dd.class` must be in the same directory as the one of the `html` file where is included.

Listing 2.5: Applet included in a web page through the HTML tag APPLET .

```
1  <html>
2    <head>
3      <title>HelloJava3Dd.java example</title>
4    </head>
5    <body>
6      <applet code="HelloJava3Dd.class" width="300"
7       height="300">
8      </applet>
9    </body>
10 </html>
```

To run the applet, the Java plug-in must be installed first to enable its execution under the Java Runtime Environment (JRE) within the browser. Installing the JRE on a computer

automatically installs the Java plug-in [36, 37]. Figure 2.5 shows a snapshot of the Java3D applet embedded in Listing 2.5. An more complex Java3D applet is shown in Figure 2.6.

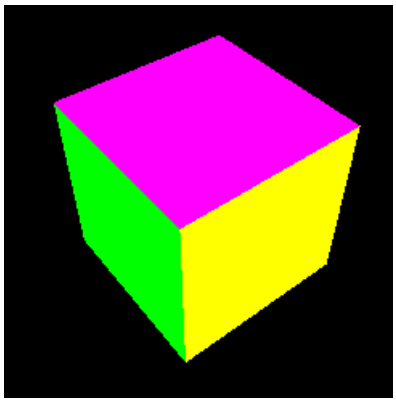

Figure 2.5: `HelloJava3Dd.class` applet running in Firefox 3.0 on Windows XP SP3. The client machine as installed the Java 2 Runtime Environment SE v1.6.0, Java 3D 1.5.1 and Java plug-in 1.6.0.

Figure 2.6: Enigma Cipher Machine applet running in Firefox 3.0 on Windows XP SP3. Retrieved from [24].

Creating an 3D desktop application in Java3D can be a relatively simple process for Java programmers. However, creating Java3D applets to embed 3D contents within a web page can become a more complex task. This requires some skills in HTML programming, as well as JavaScript programming to enhance the interactivity with the 3D web applets [36].

## 2.4 Flash 3D

The Adobe Flash Player is one of the most, if not the most, popular platforms to create interactive and visually outstanding 2D and/or 3D text, animations, web games, Rich Internet Applications (RIA), and websites for the web. Usually, in the development of Flash applications, we use the Adobe ActionScript language. This scripting language, created by Macromedia (now owned by Adobe Systems) in 1998, is based on ECMAScript (the same standard for JavaScript), and can be used for enhancing and complementing the functionalities of the Flash Player, in the same way that JavaScript enhances and complements HTML [4, 1].

In the earlier releases of the Flash Player (still from Macromedia), only some button actions and mouse interactions (through action scripting) were available, but with the release of Macromedia Flash Player 5 and with the introduction of ActionScript 1, more actions to provide better interactivity became available. It was in this Flash Player release that the ActionScript language took a JavaScript-like form, becoming a prototyped language, which allowed simple object-oriented functionalities. ActionScript 2.0 was introduced in Macromedia Flash MX 2004, or release 7 [4, 1].

This release brought two major improvements that allowed the creation of more complex actions: variable data typing and a new class syntax. With this two ActionScript releases was possible, for example, to create some complex 2D animations. ActionScript 1 and 2 use the ActionScript Virtual Machine (AVM) version 1, that is the underlying software within the Flash Player that executes ActionScript during playback [4, 1].

After the acquisition of Macromedia by Adobe, Flash was integrated in the Adobe CS3 package, and the ninth release of the Flash Player was released as Adobe Flash CS3, as well as a new version of ActionScript. With ActionScript 3, a new AVM was created (AVM2). This virtual machine only execute ActionScript 3 code, and was developed to provide gains in runtime performance and to improve developers productivity, providing resources for the creation of RIAs with audio and video streaming, as well as web games. For enhancing the performance speed, AVM2 includes a Just In Time compiler (JIT) that translates ActionScript byte-code to native machine code. ActionScript 3 introduced hardware acelleration through DirectX or OpenGL to speed up the running process of Flash in browser [4, 1].

Adobe Flash CS4 Player, or release 10, is the latest release of the Flash Platform, and also fully supports ActionScript 3. It introduced, among others, object-based animations, dynamic filters, new graphical effects, advanced text support, a new sound API, and a new drawing API, to which it was added the $z$ dimension, real perspective, textured meshes in 3D space, etc. With Flash CS4 it is now possible to perform some 3D transformations and animations, like translations and rotations of 2D surfaces on the $x$, $y$, and $z$ axes. It also allows to set up the perspective projection and camera angles to create 3D effects. Both Adobe Flash CS3 and CS4 still contains the AVM1, which executes the previous versions of ActionScript, maintaining the compatibility between the previous versions and the latest one [2].

The creation of a Flash web application, movie or game is done through the use of some authoring tools, like Adobe Flash CS4 Professional, Adobe Flex or Adobe FreeHand. Using these tools, a Flash project can be created into an `.fla` file, which contains the source ma-

terial of the Flash application. ActionScript enhancements can also be created and applied to this project using an `.as` file. Terminated a project, the `.fla` file is compiled into a non-editable and final `.swf` file. This file is ready to be manipulated, played or visualized through the Adobe Flash Player web browser plug-in. Originally `.swf` was the shorthand of "Shockwave Flash", but now stands for "Small Web Format". Today, a Shockwave file format has the `.dcr` extension and is generated by the Adobe Director authoring application. Shockwave aims at creating more complex web and multimedia applications, while Flash is used for the creation of visually appealing and interactive web interfaces, as well as web games. To interact and display a Shockwave application we use the Adobe Shockwave Player [5]. Embedding a Flash file within a web page is shown in Listing 2.6. Figure 2.7 shows the result of the Flash CS4 `.swf` file embedded in Listing 2.6.

Listing 2.6: Flash CS4 `.swf` file embedded in a web page through the HTML tag OBJECT.

```
1  <html>
2    <head>
3      <title>Flash CS4 3D Cube</title>
4    </head>
5    <body>
6
7      <object data="cs4_simple3d_cube.swf"
8      type="application/x-shockwave-flash" width="640"
9      height="440">
10     <param value="cs4_simple3d_cube.swf" />
11     </object>
12
13   </body>
14 </html>
```

Flash is now a powerful, and massive used platform, to create very visual appealing, complex, data-rich and interactive 2D and (limited) 3D contents for the web. Because of that, and due to the emerging 3D content demand, powerful 3D flash engines are emerging and evolving rapidly. 3D flash engines like Papervision3D, Sophie3D, Away3D, among others, eases the development of 3D contents for the web, using Flash and ActionScript. For a full list of Flash 3D engines and other Flash 3D resources see Appendix B. Figure 2.8 shows an interactive 3D scene within the web page completely generated with the Sophie3D flash engine.

Flash applications can also be created and displayed in mobile phones, portable electronic devices and Internet-connected digital home devices, through a lightweight version of Adobe Flash Player called Adobe Flash Lite (now on version 3) [3].

In order to display and interact with the Flash content, the Adobe Flash Player plug-in must be installed in the user web browser. While the Adobe Flash Player is available for free for most web browsers, the Adobe Flash authoring tools like Adobe Flash CS4 Professional, Adobe Flex and Adobe FreeHand are not. There are a few open source authoring

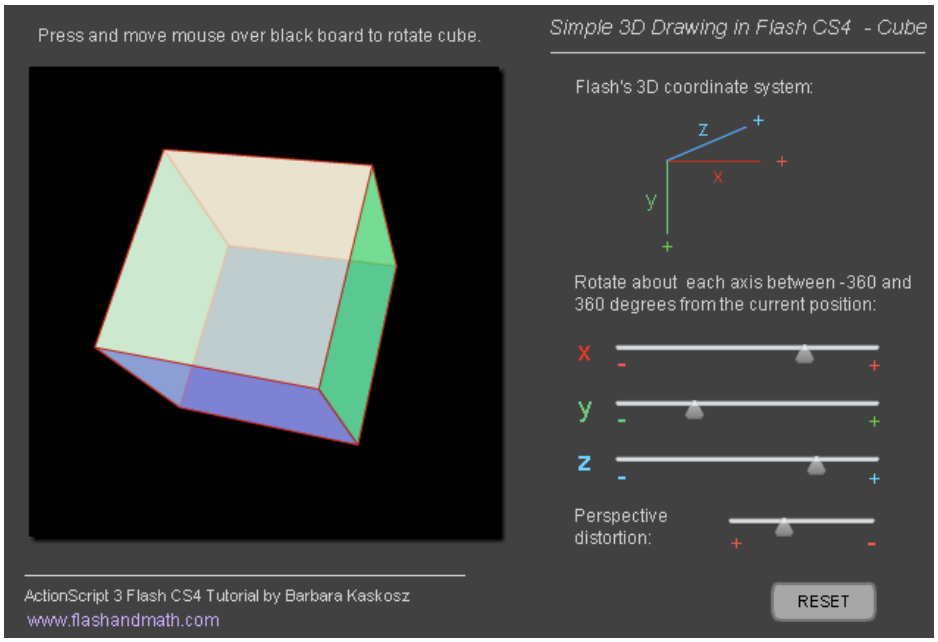


Figure 2.7: Flash CS4 Simple 3D Cube running in Windows XP SP3 with Firefox 3.0 and Adobe Flash Player 10. Flash file downloaded from [15].


Figure 2.8: Rich and interactive web 3D scene achieved through the Sophie3D flash engine. Retrieved from [13].

tools to develop Flash, but developers are forced to use the Adobe proprietary tools if they want to take full advantage of all the Flash functionalities. Flash, with its new drawing API, tries to involve non-3D developers in the creation of 3D contents, by allowing the designing in 2D and the transformation and animation in 3D. However, to create 3D contents, developers must have a solid knowledge of Flash and ActionScript. Like VRML, X3D and Java3D, Flash 3D tries to facilitate the development of 3D contents for the web, and uses a web browser plug-in for the display and interaction with these contents. But, unlike the mentioned Web3D technologies, where exists a variety of free authoring tools, Flash3D requires the use of proprietary tools to create 3D web applications. Besides, it may be a difficult development platform to use for 3D non-developers.

## 2.5 C3DL

The C3DL has been under development since 2007, and is an open source Web3D graphics library written in JavaScript that allows the creation of 3D contents in browser. It has been developed with the purpose of simplifying the creation and manipulation of 3D contents. For that, it provides a set of math, scene, and 3D objects classes to developers with little 3D programming experience. This library is built on top of the Mozilla Firefox Canvas 3D add-on. This add-on (browser specific and still experimental), provides the low level rendering functionality to C3DL, through the HTML5 canvas element (discussed in detail in Chapter 3) and an OpenGL ES 2.0 API-based rendering context written in JavaScript, allowing the display of 3D scenes and 3D objects within an area of an web page, defined by the canvas tag [27, 8].

As said before, the Canvas 3D add-on provides access to an OpenGL ES 2.0 API-based rendering context via the HTML5 canvas element. This context is called “moz-glweb20” and is a simplified and web oriented set of OpenGL ES 2.0 API-based functions. Because the “moz-glweb20” context is built on top of desktop OpenGL, users should have support for OpenGL 2.0 on the desktop if they want to take full advantage of C3DL [41, 39].

Initially, the Canvas 3D add-on provided both an OpenGL ES 1.1-based context (“moz-gles11”) and an OpenGL ES 2.0-based context (“moz-glweb20”), but due to the quick adoption of OpenGL ES 2.0 by mobile devices, and due to the complexity on maintaining the “moz-gles11” context, the currently supported context available is the “moz-glweb20” [40].

To start creating 3D contents using C3DL, the Canvas 3D add-on must be first added to the Firefox browser. Then, the C3DL API files must be downloaded. After this, developers can create and manipulate 3D scenes through JavaScript and through C3DL function calls. With C3DL it is possible to build triangular models, store and load models from COLLAborative Design Activity (COLLADA) [1] files, apply textures, apply different types of light effects (e.g. specular lighting), apply shaders (e.g. cel-shading), add collision detection and provide interaction with objects. C3DL can also be used to develop more complex 3D contents, like the creation of interactive 3D logotypes, web 3D games, particle systems, and playback motion tracking or Motion Capture (MOCAP) data [8]. The inclusion of the

[1] COLLADA is an open XML-based format with a .dae (digital asset exchange) extension, that facilitates the exchange of 3D assets between applications [20].

C3DL API in a web page is illustrated in Listing 2.7. Figure 2.9 shows the C3DL scene defined in Listing 2.7. An more complex example of C3DL can be seen in Figure 2.10.

Listing 2.7: Inclusion of the C3DL API as well of an `duck.js` script to create the 3D scene. It is also added the canvas element to the page.

```
1  <html>
2    <head>
3      <title>C3DL API Implementation</title>
4      <script type="application/javascript">
5       var SCRIPT_PATH = '../canvas3dapi/'</script>
6      <script type="application/javascript"
7      src="../canvas3dapi/c3dapi.js" ></script>
8      <script type="application/javascript" src="duck.js">
9      </script>
10   </head>
11   <body>
12     <canvas id="tutorial" width="500" height="500"></canvas>
13   </body>
14 </html>
```

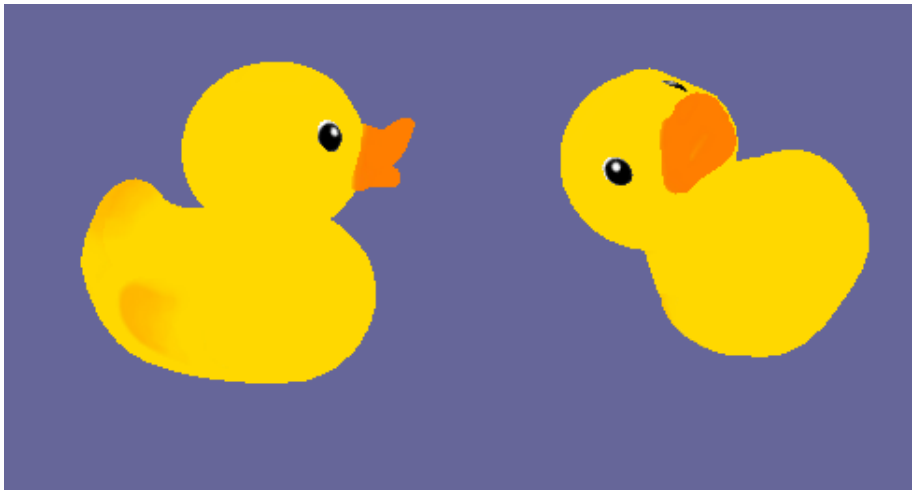

Figure 2.9: This scene shows a duck model loaded from a COLLADA file with texture applied. C3DL scene rendered using the Canvas 3D add-on 0.4.3 with Firefox 3.5 on Mac OS X v.10. Source code and model downloaded from [8].

C3DL is somehow different from the previous Web3D technologies, because even though the objective is the same, the way of providing the 3D contents is different. While all the previous Web3D technologies depend on web browsers plug-ins to interpret, interact, and display the 3D contents in browser, C3DL uses the HTML5 canvas element to render the

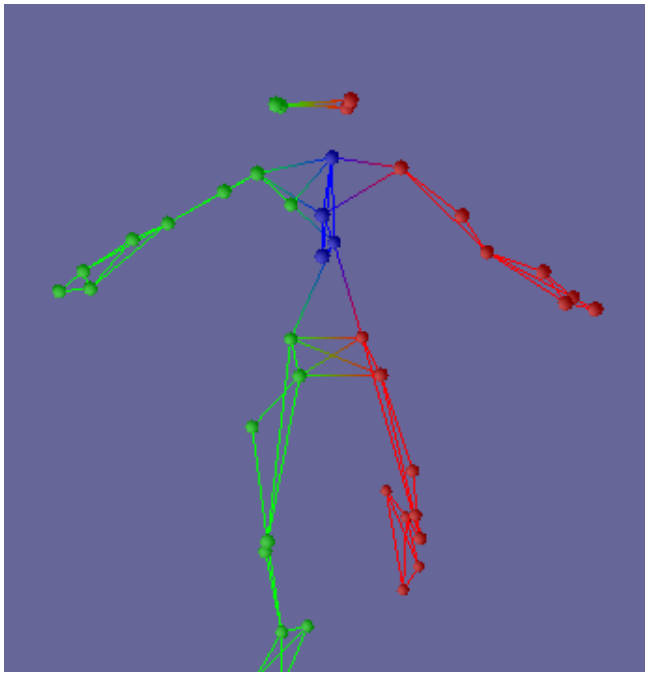

Figure 2.10: This scene shows a motion capture demo with spheres. C3DL scene rendered using the Canvas 3D add-on v.0.4.3 with Firefox 3.5 on Mac OS X v.10. Retrieved from [8].

3D contents within a specified canvas area of the Mozilla Firefox, though using the Canvas 3D add-on. This is a better way of displaying and interacting with the 3D contents, because canvas is an integral part of the web page. So, there is no need to install any third party plug-in to view and interact with the contents. However, because the Canvas 3D add-on was developed only for Firefox, users are forced to use this specific web browser (only the web browser version 3.5) to view and interact with the 3D contents. But, because it is still an experimental add-on, some issues can occur when using the library.

## 2.6 Other Web3D Technologies

VRML, X3D, Java3D, Flash 3D and C3DL are not the only 3D technologies for the web. However, they can be considered the most important ones. VRML is considered the pioneer in the development of the Web3D concept. X3D is the actual ISO standard for 3D on the web and a royalty-free 3D interchange format based on XML. Java3D provides the development of 3D contents using the powerful and widely used Java programming language. Flash 3D is the most used platform to deploy rich, interactive and visually appealing web based applications. C3DL is a new technology to display and interact with 3D contents within a web page using the canvas tag. However, there are other Web3D technologies that are worth mentioning, namely:

### 2.6.1 Ajax3D

Ajax3D stands for Asynchronous JavaScript and XML for the development of 3D contents for the web. To allow the creation of 3D web contents, it combines Ajax with X3D. Ajax allows the development of rapid, dynamic, interactive and rich web applications, while X3D provides the Scene Authoring Interface (SAI), the API that controls an X3D scene [34].

Ajax3D was developed, in 2006, to facilitate the integration and delivery of 3D contents within browser. An Ajax3D application runs in a browser, and, through JavaScript, can use the SAI to access and control a real time X3D scene, as well as the `XMLHttpRequest` to store or retrieve 3D application data, and the Document Object Model (DOM) [2] to manipulate the desired web page content. As usual with the X3D technology, and in order to parse and render the X3D scene, it is necessary the use of an X3D web browser plug-in [34]. Figure 2.11 shows an Ajax3D example.

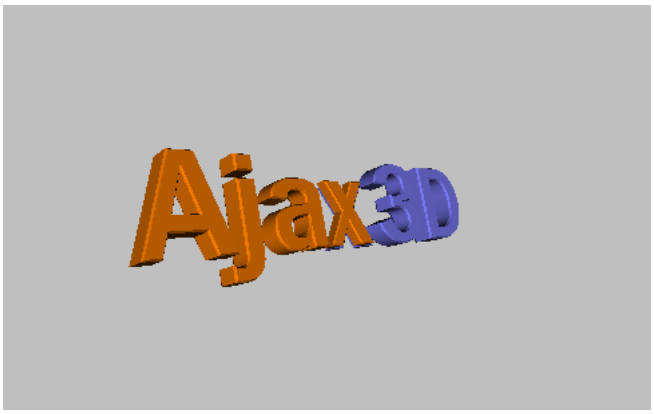


Figure 2.11: Ajax3D logotype rendered using the Vivaty plug-in v.0.9 with Internet Explorer 7 on Windows XP SP3. Retrieved from [6].

Basically, Ajax3D allows to interact with an X3D scene, and apply to or retrieve data from that scene, through JavaScript and XML [34].

### 2.6.2 3DMLW

3DMLW is an open source platform, or technology, for the creation of interactive 2D and 3D contents for the web. 3DMLW is also a technology based on XML. It has scripting support for the creation of dynamic and interactive contents, and event handling, that includes mouse, keyboard and collision events. It also allows the use of textures, lighting, shading, audio, particle engines and physics engines with collision detection. It supports `.3ds`, `.obj`, `.an8` and `.blend` file formats for 3D models. 3DMLW documents have the `.3dmlw` file extension and can be interpreted by a web browser plug-in or by a standalone 3DMLW application. This plug-in takes advantage of OpenGL for rendering purposes [35].

[2] DOM is a multi platform and language independent model for interacting with web documents [42].

3DMLW has been evolving to become a cross platform and cross browser compatible technology. By now, it is fully functional for Firefox, Safari, Opera, Chrome and Internet Explorer browsers, and for Microsoft Windows applications. The Mac OS X and Linux distributions are in beta versions and still cause problems.

To create and edit 3DMLW scenes, there is an 3DMLW editor called Quantum Hog, currently in beta version. An example of a 3DMLW scene can be seen in Figure 2.12.

Figure 2.12: 3DMLW band room scene rendered using the 3DMLW plug-in v.1.0.5 with Firefox 3.0 on Windows XP SP3. Retrieved from [35].

This technology is similar to X3D and Ajax3D, because it also encodes its contents in a XML-based file format, supports scripting to enhance interactivity, and needs a web browser plug-in to render the contents within the file. However, it is still a limited technology when compared to X3D, because X3D has more advanced graphics facilities.

### 2.6.3 O3D

O3D is a new open source JavaScript API, created by Google in 2009, for the creation of interactive and rich 3D applications in browser. This API is intended to run on multiple platforms (Windows, Mac and Linux), multiple web browsers (Firefox, Safari, Chrome and Internet Explorer), and to allow the creation of very complex and interactive web 3D applications, including virtual worlds, games, advertisements, 3D model viewers, product demos, etc. With O3D, it is also possible to import models from COLLADA files, created by Google SketchUp 6, Autodesk 3ds Max 2008 and Autodesk Maya 2008 [19].

An O3D application runs in a O3D browser plug-in. This plug-in provides hardware acceleration, advanced texturing, advanced shading capabilities and sophisticated rendering techniques. This way, O3D hides the low level graphics details to users and developers [19]. An example of an O3D application can be seen in Figure 2.13.

Figure 2.13: Earth surface with complex textures. O3D scene rendered using the O3D plug-in v.0.1 with Firefox 3.5 on Mac OS X v.10. Retrieved from [18].

This API, despite providing truly impressive 3D environments within browser, still needs the use of an web browser plug-in and it is intended for web developers with a solid background in 3D graphics. Also, the rendering of very complex and detailed 3D worlds may become very slow if the client computer does not have a good graphics card [19].

## 2.7 Summary

To conclude this chapter, it can be said that, despite the Web3D technologies, like VRML, X3D, Java3D, Flash3D, C3DL, Ajax3D, 3DMLW and O3D can provide immersive and realistic web 3D environments within browser, they all require the installation of third party web browsers plug-ins or add-ons (to take advantage of hardware acceleration), which may be a barrier to users that want to experience an easy and immediate interaction with 3D contents. Also, the creation of 3D worlds, using some of the mentioned technologies, involves learning in detail the corresponding languages, what can be difficult to non-developers or

even to web developers (see Appendix B for additional information about all the mentioned Web3D technologies). So, Glypher3D can become a useful technology to render 3D contents over the web, because it does not need the installation of any web browser plug-in or add-on. The fact that it is fully encoded in JavaScript, makes it more accessible to learn for non-developers and especially to web developers. However, it does not take advantage of hardware acceleration. Consequently, it cannot provide the level of realism and rendering speed of other Web3D technologies yet. Glypher3D only uses HTML specifications and JavaScript to provide the creation, manipulation and rendering of 3D contents within a web page. Glypher3D will be described in detail in Chapter 3.

# Chapter 3

# Glypher3D

Glypher3D is an open source Web3D API, entirely written in JavaScript, that aims at enabling the creation, manipulation and rendering of 3D contents within a web browser, without the need for installing any type of web browser plug-in or add-on. For that, we use the HTML5 canvas element together with 3D graphics algorithms encoded in JavaScript. Glypher3D attempts to provide a more intuitive way to develop and interact with the 3D contents using only JavaScript. It is a multi-platform and cross browser compatible library that runs on Windows, Mac and Linux, and within most browsers like Firefox (3+), Safari (4+), Opera (10+) and Chrome (2+). Its main drawback is the incompatibility with Microsoft Internet Explorer, but this happens so because the Internet Explorer does not support the HTML5 canvas element (used for drawing and rendering purposes).

Glypher3D can be used to create 3D scenes with points, lines, triangular or quadrangular shapes. It also allows for shading effects (e.g. flat shading), geometric transformations (e.g. rotation on the $x$-axis), hidden surface removal algorithms (e.g. back-face culling), and projection transformations (e.g. perspective projection). Additionally, it allows for wireframe and color filling objects. With Glypher3D, every object has to be created from scratch using JavaScript, because it does not allow us to import any 3D model in a specific format file. Also, it does not support texturing.

## 3.1 The Canvas Element

As said before, the HTML5 canvas element is the essential technology for Glypher3D, because it allow us to create a rectangular area within the browser, where it is possible to draw 3D scenes, without the need for installing any browser plug-in or add-on.

Basically, the HTML5 canvas element is just a HTML element that can be used to draw 2D graphics on its rendering context, and is normally used to make photo compositions, simple games, graphs, charts or animations. This element was introduced by Apple in its Mac OS X Dashboard, but it was soon adopted by the WebKit (Safari) and Gecko (Firefox) browsers. Currently, all browsers, except Internet Explorer, support this element. The canvas element is now part of the HTML5 specification [11, 23].

However, there are other technologies, like Scalable Vector Graphics (SVG), that can provide graphics for the web. SVG is an XML markup language, as well as World Wide Web Consortium (W3C) standard, for describing 2D vector graphics. It allows to draw high quality graphics (shapes) through its graphics primitives specified by XML and it has a scene DOM, what makes event handling easy (interaction with objects, e.g. mouse events). However, it is somehow difficult to integrate with HTML, has a slower rendering performance, and the process of creating graphics is more complex than with the canvas element [38].

Therefore, what led to the choice of the canvas element to provide the drawing and rendering functions to the Glypher3D library was, mainly, the simpler, faster, practical, and immediate way to draw pixel-based graphics through its 2D API. Besides, the canvas element is an integral element of the web page, which facilitates the access to its functions through JavaScript, as needed for Glypher3D. This element has a few drawbacks, namely the inexistence of a scene DOM (event handling harder) and, like SVG, it is not supported by Internet Explorer. However, some efforts are being done to make the canvas element accessible for this browser, like the Google Internet Explorer plugin called ExplorerCanvas [17]. This plugin only translates canvas commands to Vector Markup Language (VML) (XML language to develop vector graphics for Internet Explorer), because it only supports simple drawing functions and has a slower rendering process [38].

## 3.2 Client-Side Architecture

The Glypher3D architecture is illustrated in Figure 3.1. This architecture shows that Glypher-3D is a client-side JavaScript web library. This means that a client does not need to download it in order to run a Glypher3D application. If Glypher3D is included in a web page, this page, as well as all Glypher3D JavaScript files, are loaded, and run locally in the web browser of the client.

However, if web developers intend to create Glypher3D-based applications in their web pages, they must download the Glypher3D library. This library will be in a folder named "glypher3dlib", which includes the following 10 JavaScript files:

- `Glypher3D.js`.
- `Main.js`.
- `ContextRenderer.js`.
- `ShapePoints.js`.
- `Transformations.js`.
- `Perspectives.js`.
- `Illumination.js`.
- `ColorAndShading.js`.
- `DrawingPrimitives.js`.
- `UtilitiesAux.js`.

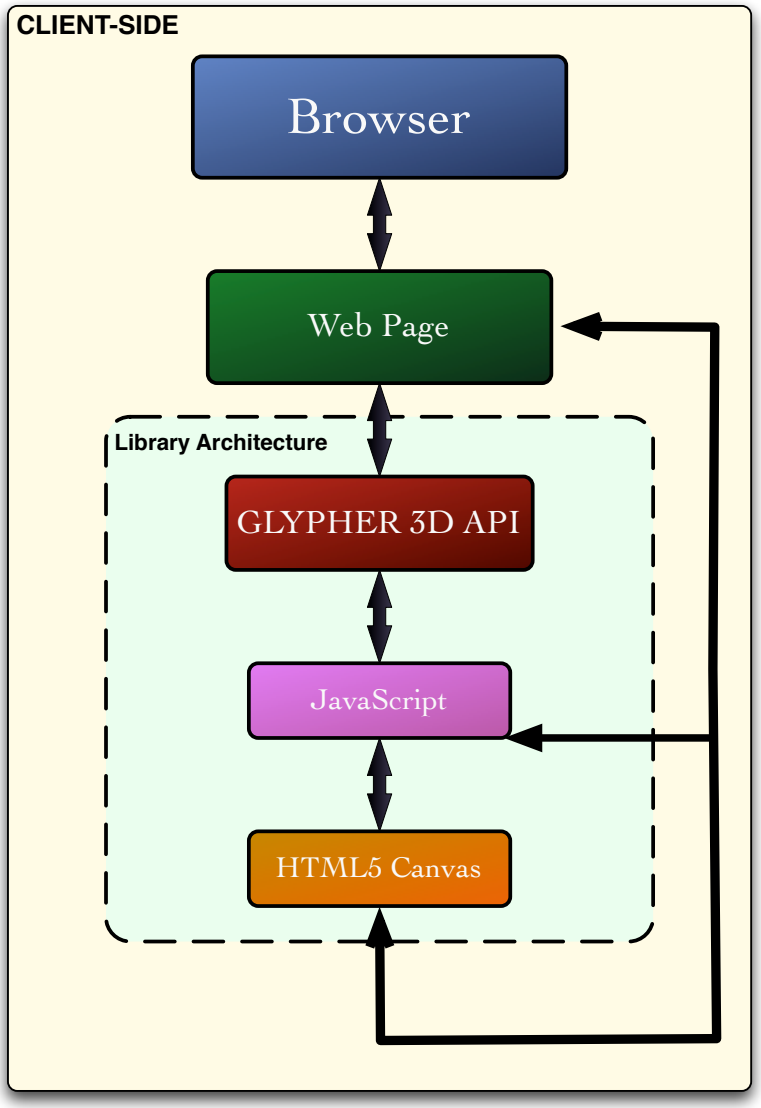


Figure 3.1: Glypher3D Client-Side Architecture.

These files contain JavaScript functions and methods, from which we can access the methods of the HTML5 canvas element. These methods are used to create the rendering context in the defined canvas area. Thus, Glypher3D takes advantage of the drawing and rendering methods of canvas. Besides, some auxiliary scripts can be created within the web page to enhance user interaction. To include Glypher3D into an web page, it is only necessary to include the `Glypher3D.js` file in the HTML file of the page (Listing 3.1).

Listing 3.1: Inclusion of Glypher3D into a web page.

```
1 <head>
2   <script type="text/javascript"
3   src="glypher3dlib/Glypher3D.js">
4   </script>
5 </head>
```

The inclusion of this JavaScript file in the HTML document sets up an association between the HTML document and all the JavaScript files of the Glypher3D library.

## 3.3 Rendering Pipeline

The Glypher3D rendering pipeline transforms a 3D scene into a 2D image on screen (canvas area). Figure 3.2 shows the diagram of the Glypher3D rendering pipeline. Its stages required to create, manipulate and render a Glypher3D scene are:

- The first stage of the Glypher3D rendering pipeline is the creation of the canvas 2D rendering context, which defines the rendering area within the web page.
- Next, it comes the time of the creation of an 3D scene, being each object incrementally build up using graphics primitives (line, line strip, line loop, triangle, triangle strip, triangle fan, quad or quad strip).
- Geometric transformations algorithms, like rotation, translation, scaling and shearing can now be applied to the created object.
- The viewer position is static and defined automatically by Glypher3D on the positive $z$-axis, though the light position can be altered by the user. The light and viewer positions are necessary for shading and hidden surface removal algorithms.
- To allow a proper view of the object and to increase the rendering speed of an Glypher3D scene, we use the painter's algorithm and the back-face culling algorithm.
- To give more realism to the scene, coloring and shading methods, like fill color, flat shading or Gouraud shading, can be applied.
- Then, to map the 3D coordinates of the 3D object onto the 2D rendering context of the canvas element, Glypher3D applies by default the parallel projection. To give the 3D look to the scene can be applied the perspective projection.
- The final stage of the Glypher3D rendering pipeline is the display. This step allows to draw and output the 3D scene, filled or wire-framed onto the 2D canvas rendering context.

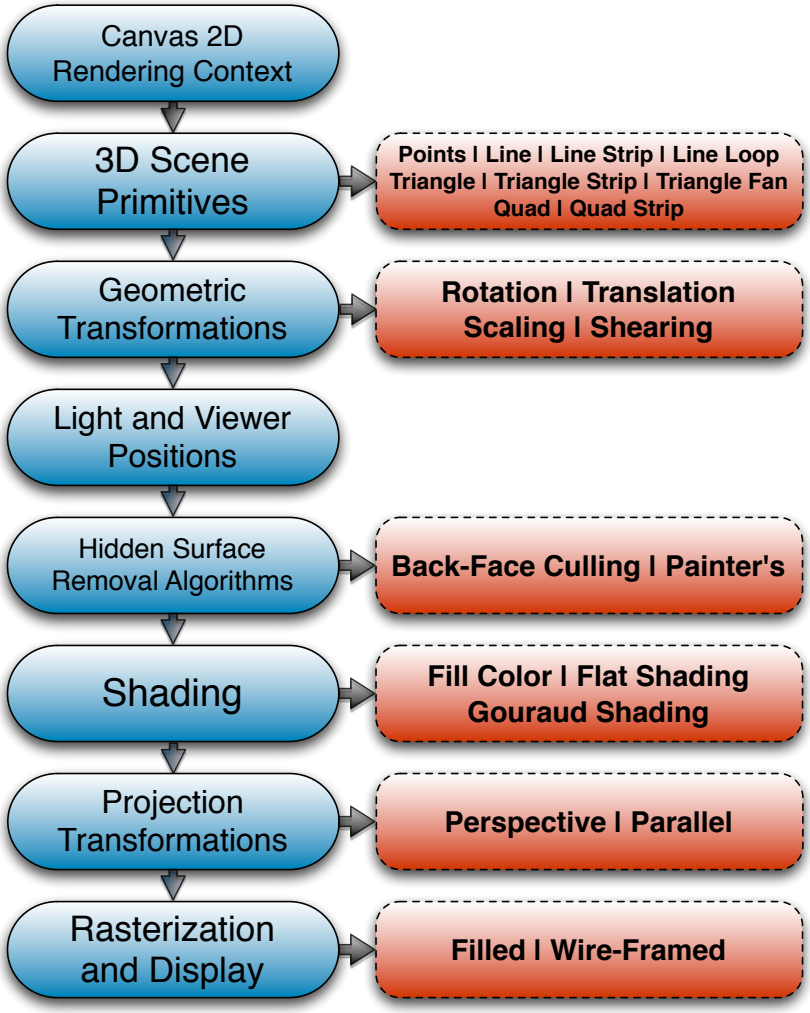


Figure 3.2: Glypher3D Rendering Pipeline.

As shown in Figure 3.2, the Glypher3D rendering pipeline is somehow simpler (has less steps) than the classic rendering pipeline. This happens, because the Glypher3D rendering pipeline is intended to provide more usability and to allow the render of 3D scenes in a more intuitive and faster way. Therefore, some steps of the classic rendering pipeline were discarded due to avoid extra and unnecessary calculations. The steps discarded were the clipping and coordinate normalization. The clipping process cuts the objects that are outside the rendering (volume) area. In Glypher3D, the positioning of the objects within the rendering area is up to the developer, and if the object is placed or moved outside the rendering area, it maintains its geometry and can be placed again within the rendering area. The normalization process of coordinates consists in ensuring that the the 3D scene (regardless of its size) will fit into the 2D rendering area. In Glypher3D, the developer must

create objects within the rendering area. The full and detailed explanation of the Glypher3D rendering pipeline stages will be described in the next sections.

## 3.4 Creation of the Canvas 2D Rendering Context

As shown in Figure 3.2, the creation of a canvas 2D rendering context is the first stage of the rendering pipeline. This is done by adding the HTML canvas tag to the body of the web page, as shown in Listing 3.2. The canvas tag is used to define a rectangular region of pixels (i.e. a viewport or canvas), where the 3D contents will be output to.

Listing 3.2: Canvas tag added on web page.

```
1 <body>
2   <canvas id="area" width="600" height="600"></canvas>
3 </body>
```

Listing 3.2 shows that the canvas tag has a string identifier (id="area"), a width and height parameters. The string identifier is necessary to get access to the canvas tag. It is not specific to this tag, but it is the default HTML parameter that can be applied to almost every HTML tag. The width and height parameters are optional, and serve to define the size of the canvas area in pixels. If no width and height are specified, the canvas area is created with the default size, which is 300 pixels wide and 150 pixels high. The size of the canvas area can also be defined using an Cascading Style Sheet (CSS) file. Optional HTML style attributes as, for example, border size and background color can be applied to the canvas tag. These attributes do not affect in any way the rendering of Glypher3D objects onto the canvas area. By default, the background color of the canvas area is white [11].

After adding the canvas tag to the web page, this tag become accessible through its identifier, i.e. id="area". Accessing to the canvas tag and retrieval of the canvas DOM node is done as in Listing 3.3.

Listing 3.3: Retrieval of the canvas DOM node using the `getElementById` method.

```
1 <script type="text/javascript">
2   var canvas = document.getElementById('area');
3 </script>
```

Now, it is possible to access to the 2D rendering context using the canvas `getContext` method. The access to the rendering context is done using the following Glypher3D method:

- GLY_INIT_2D(canvas DOM node).

This method must be used as in Listing 3.4.

Listing 3.4: Creation of the canvas 2D rendering context.

```
<script type="text/javascript">
  GLY_INIT_2D(canvas);
</script>
```

This method creates a global `CanvasRenderingContext2D` object as needed to render scenes on the rendering area. A full example that illustrates the creation of a canvas 2D rendering context, as a starting point to create, manipulate and display 3D contents using Glypher3D, is shown in Listing 3.5.

Listing 3.5: Creation of the canvas 2D rendering context within a web page.

```
<html>
  <head>
    <title>Include "canvas" on page</title>
    <script type="text/javascript"
    src="glypher3dlib/Glypher3D.js">
    </script>
    <script type="text/javascript">
      var canvas = document.getElementById('area');
      GLY_INIT_2D(canvas);
    </script>
  </head>
  <body>
    <canvas id="area" width="600" height="600"></canvas>
  </body>
</html>
```

By default, the canvas 2D rendering context represents a discrete coordinate system (i.e. grid of pixels) whose origin (0,0) is at the top left corner, with the positive $y$-axis pointing down. In Glypher3D, the origin of this coordinate system is considered to be at the center, with the positive $x$-axis pointing right and the positive $y$-axis pointing up, as illustrated in Figure 3.3.

To change the default canvas $y$-axis orientation (down) to the standard orientation (up), we internally invert the signal of the $y$-coordinates of all vertices used in the scene. To change the origin of the default coordinate system, from the top left corner to the center of the canvas area, we internally add the values $width/2$ and $height/2$ to the x- and y-coordinates of every vertex used in the scene. This process is done before rendering the scene.

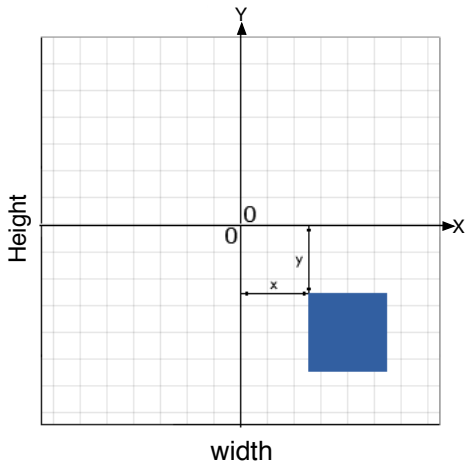


Figure 3.3: Glypher3D 2D coordinate system.

## 3.5 Creation of the 3D Scene

The second stage of the Glypher3D rendering pipeline has to do with 3D scene modeling. Building up a 3D scene means to define the geometry of its constituent objects. Similar to OpenGL, the geometry of a 3D object is defined by Glypher3D graphics primitives.

### 3.5.1 Glypher3D 3D Coordinate System

When building a 3D scene in Glypher3D, we must be aware of the 3D coordinate system. This coordinate system adds a third dimension, the $z$-coordinate axis. This 3D coordinate system is right-handed. This means that the origin (0,0,0) is at the center of the rendering area, the positive $x$-axis points right, the positive $y$-axis points up and the positive $z$-axis points backwards, as shown in Figure 3.4.

Glypher3D provides a set of drawing primitives similar to those found in [7, 14] for creating shapes within a web page. Next, we will describe such graphics primitives.

### 3.5.2 Points

Every object in 3D is built using points called vertices. However, points and vertices are different concepts. Points are just random positions in 3D space, while vertices are positions that define an object (e.g. the vertices of a triangulated mesh). A point is defined by a triple $x$, $y$ and $z$ coordinates in the Glypher3D coordinate system (Figure 3.4). A point is generated by using the following Glypher3D primitive:

- GLY_POINT_3D(float x, float y, float z);.

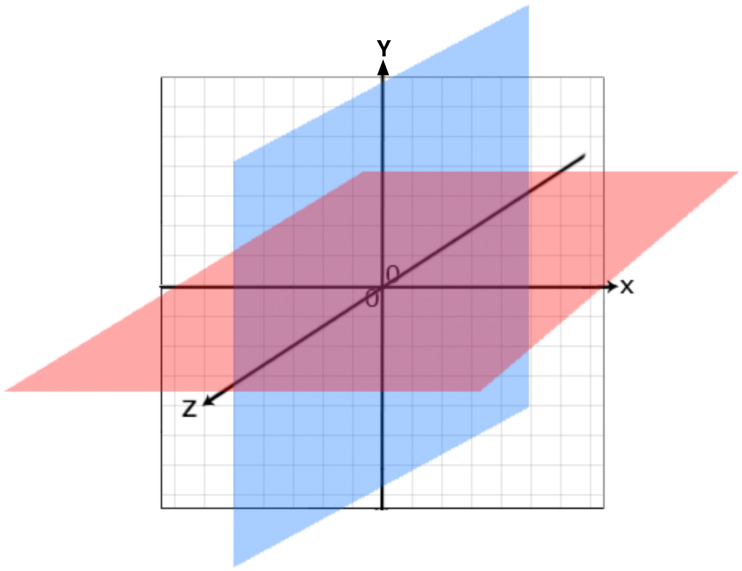


Figure 3.4: Glypher3D right-handed 3D coordinate system.

This primitive is encoded as a JavaScript class, so it allows the production of point objects as indicated in Listing 3.6.

Listing 3.6: Creation of three 3D point objects.

```
1 <script type="text/javascript">
2    var point1 = new GLY_POINT_3D(0,0,100);
3    var point2 = new GLY_POINT_3D(100,-100,100);
4    var point3 = new GLY_POINT_3D(-100,-100,100);
5 </script>
```

When the points are associated to a shape, they become the vertices of that shape, as demonstrated in sequel.

### 3.5.3 Shape

In Glypher3D, a shape is the same as an object, a model or a polygon. A shape can be a line with two vertices, or a complex model with a number of triangles. It is compounded by two parameters: the constituent points (vertices of the shape) and the identifier that denotes the type of shape. There are several shape identifiers, namely:

- "GLY_LINE"
- "GLY_LINE_STRIP"
- "GLY_LINE_LOOP"

- "GLY_TRIANGLE"
- "GLY_TRIANGLE_STRIP"
- "GLY_TRIANGLE_FAN"
- "GLY_QUAD"
- "GLY_QUAD_STRIP"

The vertices of the shape are the first parameter of the Glypher3D shape class, and have to be inserted as an array of points. The second parameter specifies the type of shape, a string, i.e. the shape identifier. Note that, internally, the vertices in the array are all ordered in a clock-wise order for shading purposes. To create an shape object, it is used the following Glypher3D shape class:

- GLY_SHAPE(array of points, primitive string).

The Glypher3D shape class must be coded as in Listing 3.7.

Listing 3.7: Creation of a triangle shape object, using the points created in Listing 3.6.

```
1 <script type="text/javascript">
2   var triangle = new GLY_SHAPE(new Array(point1,point2,point3)
3   ,"GLY_TRIANGLE");
4 </script>
```

Figure 3.5 illustrates how to use all the Glypher3D primitives. The line strip, line loop, triangle strip, triangle fan, and quad strip primitives are just to avoid the duplication of shared vertices. But, internally, all line strips and line loops are transformed into lines, all triangle strips and triangle fans are transformed into triangles and all the quad strips are transformed into quads, meaning that Glypher3D will draw redundant vertices and edges when using the mentioned primitives. This is done to ease the drawing process and not to slow the rendering performance. Next, we will describe the shape identifiers.

### 3.5.4 Lines

In 3D graphics, lines are considered as line segments, and there is not the concept of infinite lines as it happens in mathematics. The endpoints of an line are defined by vertices. Just like in OpenGL, it is possible to create three types of line primitives, whose identifiers are the following:

- "GLY_LINE".
- "GLY_LINE_STRIP".
- "GLY_LINE_LOOP".

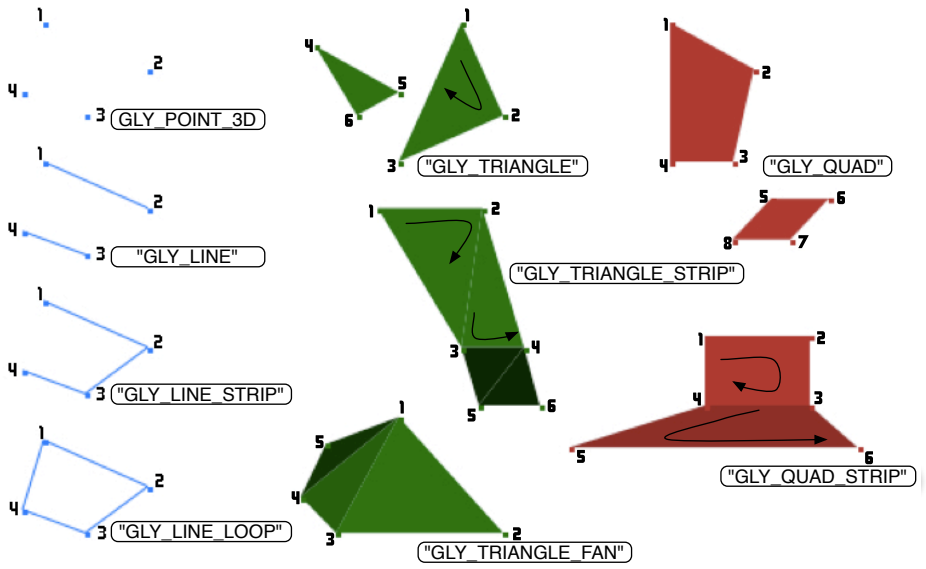


Figure 3.5: Glypher3D graphics primitives.

As mentioned above, the generation of a specific line primitive requires the production of an object from the ”GLY_SHAPE” class, by invoking the respective class constructor method with the identifier of such a line primitive, passed as one of the parameters. The production of an shape object is illustrated in Listing 3.7.

The ”GLY_LINE” identifier is used to create a straight line segment defined by two endpoints. The ”GLY_LINE_STRIP” is used to create an open polyline from a sequence of at least two points. The ”GLY_LINE_LOOP” identifier allows to create a closed polyline from a sequence of at least three points.

The line strip and line loop allow us to reduce the number of vertices used to create lines with common vertices. For example, if we want to create two lines, where the last vertex of the first line is the same as the first vertex of the second line, we can use the line strip using just three vertices instead of four vertices.

### 3.5.5 Triangles

As known, a polygon can be decomposed into a collection of triangles, creating what is usually referred to as a mesh. If a mesh is only compounded by triangles, these triangles are also called facets. As with OpenGL, Glypher3D supports three primitives related to triangles, whose identifiers are:

- ”GLY_TRIANGLE”.

- ”GLY_TRIANGLE_STRIP”.
- ”GLY_TRIANGLE_FAN”.

Again, a triangle primitive is generated using the ”GLY_SHAPE” method with the corresponding identifier as parameter. ”GLY_TRIANGLE” is the identifier used to create a triangle from three points. ”GLY_TRIANGLE_STRIP” is the identifier required to create a strip of triangles from a sequence of at least four points. The ”GLY_TRIANGLE_FAN” identifier allows for the creation of a fan of triangles from a sequence of at least four points.

The triangle strip and triangle fan are intended to avoid the duplication of shared triangle vertices. For example, to draw two connected triangles sharing an edge, we only need to generate a triangle fan with four vertices, instead of six vertices.

### 3.5.6 Quads

Quad, or quadrangle, is the other polygon type supported by Glypher3D. Similar to OpenGL, Glypher3D supports two quads primitives. They have the following identifiers:

- ”GLY_QUAD”.
- ”GLY_QUAD_STRIP”.

As usual, a quad primitive is generated using the ”GLY_SHAPE” method with the corresponding identifier as parameter. The ”GLY_QUAD” identifier allows the creation of square objects given four points, while the ”GLY_QUAD_STRIP” identifier allows to create a strip of square objects from a sequence of at least six points. It is only possible to form quad strips with an even number of vertices.

The quad strip is also intended to avoid the duplication of shared vertices. For example, to draw two connected squares sharing an edge, it is only necessary to generate six vertices, instead of eight vertices.

### 3.5.7 Scene

As shown in Figure 3.6, a Glypher3D scene is compounded by the light position, viewer position, background color and shapes. This means that any change to the background color or light position will be equally applied to all shapes in the scene. On the other hand, a shape is compounded mainly by geometric primitives (created points and graphics primitives). Each shape object is defined independently, which means that any following transformation, hidden surface removal algorithms, color and shading algorithms, projections, or the display, are general methods that are applied to each shape individually. With Glypher3D it is possible to create multiple shapes to form an 3D scene.

## 3.6 Geometric Transformations

Geometric transformations, also called affine transformations, are useful as modeling tools for objects and scenes [29]. These transformations allow us to change the position, orienta-

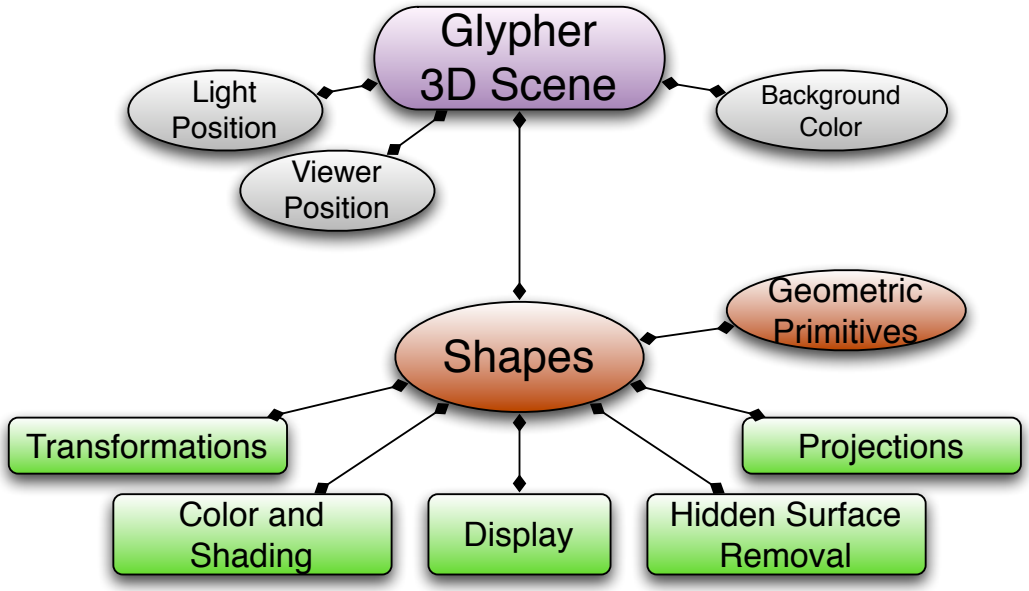


Figure 3.6: Glypher3D scene composition.

tion, size, and shape of the 3D objects through transformation matrices. With them, we can rotate an object, translate an object to a new location, or to change the size of an object.

### 3.6.1 Rotation

The rotation allows us to change the orientation of an 3D object. The Glypher3D rotation matrices allow us to rotate an object around the $x$, $y$ and $z$ axes, given an angle in degrees.

If we want to rotate an object around the $x$ axis, we only change the $y$ and $z$ coordinates. Thus, we do not change the $x$ coordinate, as in the equation system (3.1).

$$\begin{cases} x' = x \\ y' = \cos\theta.y - \sin\theta.z \\ z' = \sin\theta.y + \cos\theta.z \end{cases} \tag{3.1}$$

This equation system translates into the following matrix form:

$$\begin{bmatrix} x' \\ y' \\ z' \\ 1 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & \cos\theta & -\sin\theta & 0 \\ 0 & \sin\theta & \cos\theta & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix} \tag{3.2}$$

The $x$, $y$ and $z$ values, correspond to the initial object position. The $x'$, $y'$ and $z'$ values, correspond to the final object position after a rotation around the $x$ axis.

Similar to the rotation around the $x$ axis, to rotate an object around the $y$ axis, we only change the $x$ and $z$ coordinates, as expressed in the equation system (3.3).

$$\begin{cases} x' = \cos\theta.x + \sin\theta.z \\ y' = y \\ z' = -\sin\theta.x + \cos\theta.z \end{cases} \tag{3.3}$$

This equation system translates into the following matrix form:

$$\begin{bmatrix} x' \\ y' \\ z' \\ 1 \end{bmatrix} = \begin{bmatrix} \cos\theta & 0 & \sin\theta & 0 \\ 0 & 1 & 0 & 0 \\ -\sin\theta & 0 & \cos\theta & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix} \tag{3.4}$$

Similarly to the previous two rotations, to rotate an object around the $z$ axis, we only change the $x$ and $y$ coordinates, as expressed in the equation system (3.5).

$$\begin{cases} x' = \cos\theta.x - \sin\theta.y \\ y' = \sin\theta.x + \cos\theta.y \\ z' = z \end{cases} \tag{3.5}$$

This equation system translates into the following matrix form:

$$\begin{bmatrix} x' \\ y' \\ z' \\ 1 \end{bmatrix} = \begin{bmatrix} \cos\theta & -\sin\theta & 0 & 0 \\ \sin\theta & \cos\theta & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix} \tag{3.6}$$

The Glypher3D rotation methods around the $x$, $y$, and $z$ axes are, respectively:

- GLY_ROTATION_X(float angle).
- GLY_ROTATION_Y(float angle).
- GLY_ROTATION_Z(float angle).

Listing 3.8 illustrates the rotation of a triangle, around each coordinate axes, being $\theta = 0.8$ the rotation angle.

Listing 3.8: All three rotations applied to the triangle shape created in Listing 3.7.

```
1 <script type="text/javascript">
2   triangle.GLY_ROTATION_X(0.8);
3   triangle.GLY_ROTATION_Y(0.8);
4   triangle.GLY_ROTATION_Z(0.8);
5 </script>
```

### 3.6.2 Translation

The translation allow us to change the position of an object, and to place the object in space. Every positional change is a composition of changes along the coordinates axes. Thus, the translation can be expressed through the equation system (3.7).

$$\begin{cases} x' = x + \Delta_x \\ y' = y + \Delta_y \\ z' = z + \Delta_z \end{cases} \tag{3.7}$$

In equation system (3.7), the $\Delta_x$, $\Delta_y$, and $\Delta_z$ stand for the displacement along each coordinate axis, $x$, $y$, $z$, respectively.

This equation system translates into the following matrix form:

$$\begin{bmatrix} x' \\ y' \\ z' \\ 1 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & \Delta_x \\ 0 & 1 & 0 & \Delta_y \\ 0 & 0 & 1 & \Delta_z \\ 0 & 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix} \tag{3.8}$$

The Glypher3D translation method is:

- GLY_TRANSLATION(float x, float y, float z).

In Listing 3.9, we have a translation of a triangle by $\Delta_x = -100$, $\Delta_y = 0$, and $\Delta_z = 0$.

Listing 3.9: Translation applied to the triangle shape created in Listing 3.7.

```
1 <script type="text/javascript">
2     triangle.GLY_TRANSLATION(-100,0,0);
3 </script>
```

### 3.6.3 Scaling

The scaling transformation allow us to increase or decrease the original size of the object along the $x$, $y$, and $z$ axes. Let us denote $\lambda x$, $\lambda y$, and $\lambda z$ as the scaling factors along the $x$-, $y$-, and $z$-axis, respectively. Dilating an object requires that at least one of the scaling factors is greater than 1. On the contrary, a contraction is performed using a positive scaling factor lesser than 1.

The scaling transformation can be expressed through the equation system (3.9).

$$\begin{cases} x' = x.\lambda_x \\ y' = y.\lambda_y \\ z' = z.\lambda_z \end{cases} \tag{3.9}$$

This equation system translates into the following matrix form:

$$\begin{bmatrix} x' \\ y' \\ z' \\ 1 \end{bmatrix} = \begin{bmatrix} \lambda_x & 0 & 0 & 0 \\ 0 & \lambda_y & 0 & 0 \\ 0 & 0 & \lambda_z & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix} \tag{3.10}$$

The Glypher3D scaling method is:

- GLY_SCALING(float x, float y, float z).

In Listing 3.10, the triangle has been applied a non-uniform scaling with factors $\lambda_x = 0.5$, $\lambda_y = 0.8$, and $\lambda_y = 1$.

Listing 3.10: Scaling applied to the triangle shape created in Listing 3.7.

```
1 <script type="text/javascript">
2   triangle.GLY_SCALING(0.5,0.8,1);
3 </script>
```

### 3.6.4 Shearing

In Glypher3D, the shearing transformation is performed along the $x$, $y$, and $z$-axes. Shearing distorts the shape and changes the position of the object along the coordinate axes. The $\kappa_x$, $\kappa_y$, and $\kappa_z$ are the shearing factors along the $x$, $y$ and $z$-axes.

The shearing transformation can be expressed through the equation system (3.11).

$$\begin{cases} x' = x + (\kappa_x.y) + (\kappa_x.z) \\ y' = y + (\kappa_y.x) + (\kappa_y.z) \\ z' = z + (\kappa_z.x) + (\kappa_z.y) \end{cases} \tag{3.11}$$

This equation system translates into the following matrix form:

$$\begin{bmatrix} x' \\ y' \\ z' \\ 1 \end{bmatrix} = \begin{bmatrix} 1 & \kappa_x & \kappa_x & 0 \\ \kappa_y & 1 & \kappa_y & 0 \\ \kappa_z & \kappa_z & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix} \tag{3.12}$$

The Glypher3D shearing method is:

- GLY_SHEARING(float x, float y, float z).

In Listing 3.11, the triangle is distorted along the $x$-axis with a shearing factor of $\kappa_x = 0.8$, $\kappa_y = 0$ and $\kappa_z = 0$.

Listing 3.11: Shearing applied to the triangle shape created in Listing 3.7. Shape sheared in the $x$ axis.

```
1 <script type="text/javascript">
2   triangle.GLY_SHEARING(0.8,0,0);
3 </script>
```

## 3.7 Light and Viewer Positions

In 3D, we must define the positions of viewer and light. This is mandatory for back-face culling, as well as for illumination purposes (flat shading and Gouraud shading). In Glypher3D, the viewer position is always static, defined by default on the positive $z$-axis and pointing in a negative direction towards the center (0,0,0) of the canvas area. It is essential for the back-face culling and Gouraud shading. On the other hand, the position of the light in a scene is mandatory for illumination and shading algorithms (flat shading and Gouraud shading), as needed to provide realistic scenes. In short, the viewer and light are constituent objects of a scene. The light and viewer positions influence in the same manner all the objects within an Glypher3D scene.

The final position of these vectors is always the center of the canvas area (0,0,0), and it is only possible the change the initial position of the light vector. The viewer vector is always in the z-axis. For example, by changing the light position to the left, while applying a flat shading algorithm, the faces of the object will be brighter on the left side and darker on the opposite side.

The light position method is:

- GLY_LIGHT_POSITION(float x, float y, float z).

To apply this method to an Glypher3D scene, we proceed as in Listing 3.12.

Listing 3.12: Changes on the light vector position applied to an Glypher3D scene.

```
1 <script type="text/javascript">
2   GLY_LIGHT_POSITION(-100,0,100);
3 </script>
```

An example of the light vector in action can be seen in Figure 3.7.

## 3.8 Hidden Surface Removal Algorithms

Glypher3D implements two hidden surface removal algorithms: the painter's algorithm and the back-face culling algorithm. Although the painter's algorithm does not remove any face of the object, we have included it in this category because it deals with the face positions of the objects in the scene. It can be considered as a depth sorting algorithm. These algorithms

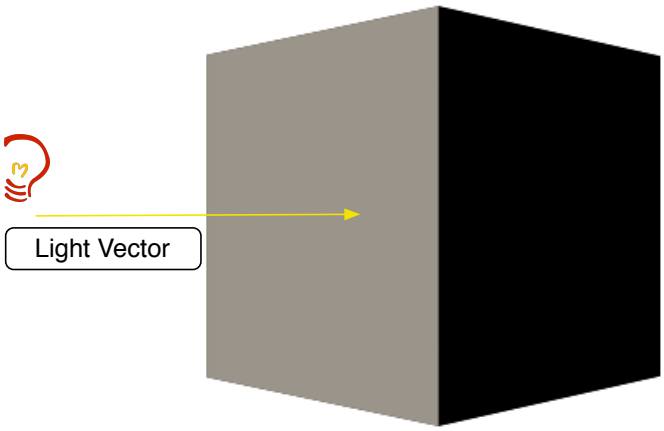


Figure 3.7: Glypher3D light position.

have the common objective of providing a correct and faster rendering of the objects within a Glypher3D scene.

### 3.8.1 Painter's Algorithm

This algorithm was implemented to allow a proper view of the objects, because without it, visibility problems can occur when displaying the objects. For example, if one creates an object or shape from front to back, i.e. creating the closest faces first and then the furthest ones, the colors of the furthest faces can overlap the colors of the closest ones. Applying this algorithm solves this potential problem of color overlapping. For that, it reorders the faces of objects so that they can be drawn from back to front as an oil painting artist does. It reorders the faces through their centroid or barycentric positions. Centroid or barycenter means the center of an object face. Because, in Glypher3D, it is possible to create triangular and quadrangular shapes, the centroid $C$ is calculated differently for these shapes. For triangles, it is used Equation (3.13), where $z_1$, $z_2$, and $z_3$ stand for the $z$ coordinate of each vertex.

$$C = (z_1 + z_2 + z_3)/3 \tag{3.13}$$

For quadrangles, it is used Equation (3.14), where $z_1$, $z_2$, $z_3$ and $z_4$ also stand for the $z$ coordinate of each vertex.

$$C = (z_1 + z_2 + z_3 + z_4)/4 \tag{3.14}$$

For each face, we sum up the $z$ coordinates of all vertices that compound that face, and by dividing that sum by the number of vertices of the face (three for triangles, and four for quadrangles), we obtain the centroid. Then, the faces are reordered from the ones with

the smallest centroids to the biggest ones, and the furthest faces are rendered first than the closest ones.

The Glypher3D method for the painter's algorithm is:

- GLY_PAINTERS_ALGORITHM().

The use of this method is illustrated in Listing 3.13.

Listing 3.13: Painter's algorithm applied to a quad shape.

```
1 <script type="text/javascript">
2   quads.GLY_PAINTERS_ALGORITHM();
3 </script>
```

Figure 3.8, shows the result of rendering an object without the painter's algorithm (on the left) and with such algorithm (on the right).

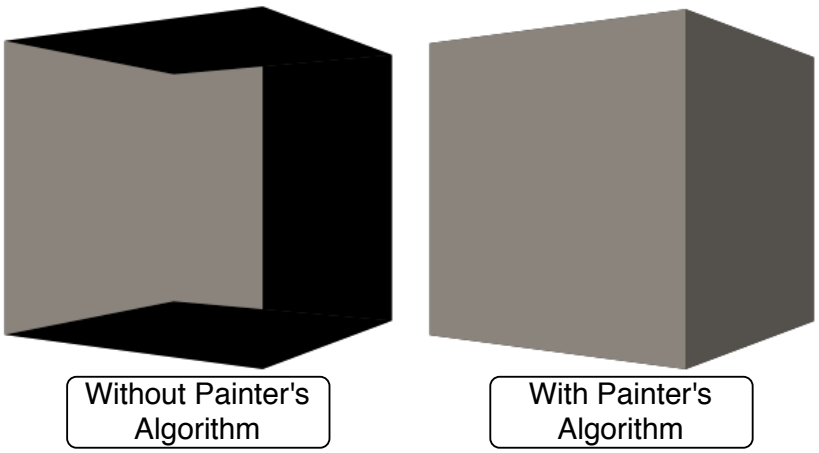


Figure 3.8: Glypher3D painter's algorithm. On the left, a cube without the algorithm, on the right, a cube with the algorithm.

Notice that the painter's algorithm only reorders the objects faces by their centroids. It does not reorder the positions of the objects within an Glypher3D scene. The order by which the objects are placed in a scene is up to the developer.

### 3.8.2 Back-Face Culling Algorithm

The objective of this algorithm is to determine which faces, well as edges, of the objects in a scene are visible from the center of projection, i.e. from the viewer position, removing then the hidden edges and faces from the rendering process. The removal of these hidden edges and faces alleviates the render processing load because we only need to render the

geometry that is visible to the viewer. This is particularly useful for computation-intensive equations as those used, for example, in Gouraud shading to color the visible faces.

The back-face culling algorithm determines which faces are hidden to the viewer, discarding them from rendering afterwards. This algorithm consists of the following steps:

- *To calculate the face normals.* The calculation of a face normal takes into account the order by which its vertices are created. In Glypher3D, the vertices must be arranged in a clock-wise way. The normal to a face is just given by the cross product of two vectors, $v_1 = B - A$ and $v_2 = C - A$, where $A$, $B$, and $C$ are three consecutive vertices in the boundary of a face, as illustrated in Figure 3.9.

- *To determine the visibility of faces.* This is done using the dot product of each face normal with the viewer vector. The viewer vector is defined implicitly in the positive $z$-axis towards the origin, because this eases the computations. If the face normal and the viewer vector point to each other with an angle of $\theta = 180$ degrees, the face is fully visible. When the face normal points away from the viewer position, i.e. when they point to the same side with an angle of $\theta = 0$ degrees, the face is not rendered. Therefore, if the dot product is negative, the face is visible, otherwise it is not visible.

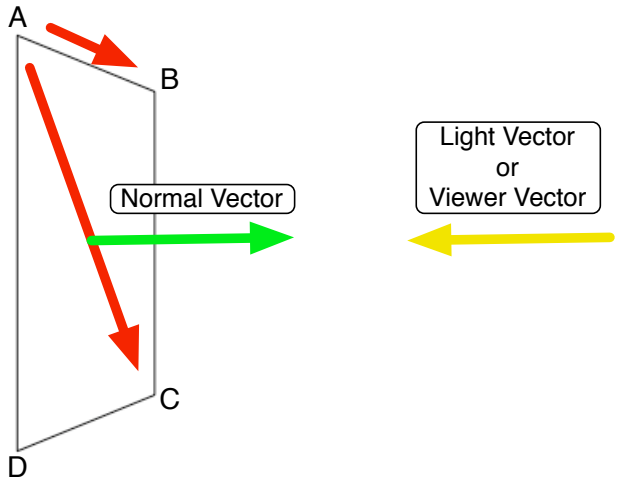


Figure 3.9: Surface normal vector on a quad shape, as well as the vectors created with three of the four vertices.

The Glypher3D method for the back-face culling algorithm is:

- GLY_BACKFACE_CULLING().

Applying the previous method to a quadrangle is done as shown in Listing 3.14.

Listing 3.14: Back-face algorithm applied to a quad shape.

```
1 <script type="text/javascript">
2     quads.GLY_BACKFACE_CULLING();
3 </script>
```

Figure 3.10, shows an object on the left, without the back-face culling algorithm, and on the right, with this algorithm.

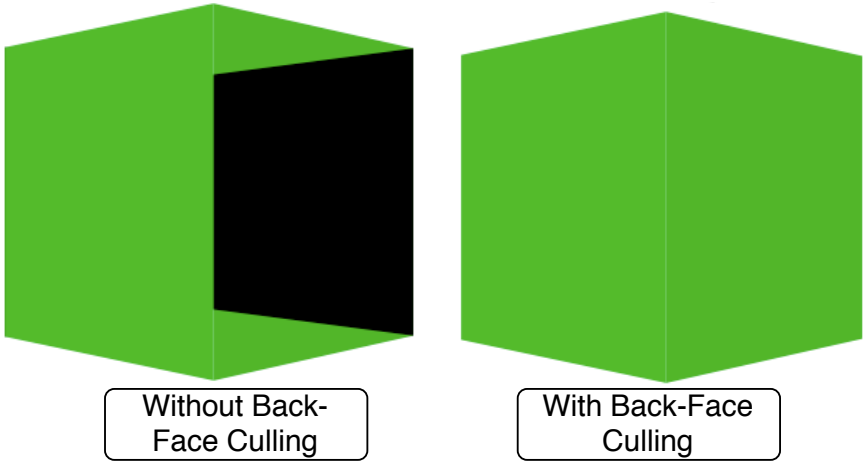


Figure 3.10: Glypher3D back-face culling algorithm. On the left, a cube without the algorithm, on the right, a cube with the algorithm.

## 3.9 Coloring and Shading

The coloring and shading methods are essential to give more realism to the scene, making it more visual appealing and rewarding. These methods use the R (red), G (green) and B (blue) color model to color the polygonal faces of each object in a given scene. Each RGB component must be defined with an integer value between 0 and 255. The Glypher3D coloring and shading methods take advantage of some of the HTML5 canvas element attributes, i.e. coloring, styling and pixel-based manipulation methods [23]. With Glypher3D, it is possible to change the background color of the scene, change the color of shapes and their constituents, and apply the flat shading algorithm or the Gouraud shading algorithm.

### 3.9.1 Background Color

The background color of the canvas area can be changed through the HTML `backgroundColor` style attribute of the canvas tag. This attribute allow us to change the default color (white) of the canvas area. So, to access this canvas style attribute and, ultimately, to change the color of the scene, we use the following Glypher3D method:

- GLY_BACKGROUND_COLOR(canvas DOM node, int R, int G, int B).

This method has as parameters the canvas DOM node variable and the RGB values. The first parameter allow us to get access to the properties of the canvas tag, being the RGB values used to assign a new color to the canvas area. Listing 3.15 illustrates this.

Listing 3.15: New background color applied to the scene.

```
<script type="text/javascript">
  var canvas = document.getElementById('area');
  GLY_BACKGROUND_COLOR(canvas,100,100,200);
</script>
```

In Figure 3.11, we see the background with the color (100,100,200).

### 3.9.2 Wireframe Color

Glypher3D allows for building and coloring wireframe objects. This is done using the following method:

- GLY_WIREFRAME_COLOR(int R, int G, int B, float A).

In addition to the RGB values, this method has a fourth parameter, the alpha parameter $A$, to change the transparency of the edges. This value must be between 0 and 1. This method takes advantage of one of the HTML5 canvas element style attributes to save the colors of the object edges. The canvas style attribute that represents the color of edges is:

- `canvas context.strokeStyle = "rgba(R,G,B,A)"`.

The `strokeStyle` is a method of the canvas rendering context. This method is used to save the colors of the object edges [23]. These colors will only be applied when we intend to draw the wireframe objects. Figure 3.11 shows a wireframe object on the left, which was colored as indicated in Listing 3.16.

Listing 3.16: Change of the color of the object edges.

```
<script type="text/javascript">
  quads.GLY_WIREFRAME_COLOR(255,0,0,1);
</script>
```

With this method, all edges of the same object will have the same color, i.e. it is not possible to color edges of the same object with different colors.

### 3.9.3 Fill Color

The fill color method is the most simple and less realistic way of coloring an object, because it applies the same color to all faces of an object. It does not take into account the light position. To fill all faces with a given color, we use the following method:

- GLY_FILL_COLOR(int R, int G, int B, float A).

It is also possible to define the transparency of the object with the alpha parameter. This facility takes advantage of an HTML5 canvas element style attribute to save the object color. The canvas style attribute that represents the color to use inside the object is:

- `canvas context.fillStyle = "rgba(R,G,B,A)".`

This canvas rendering context method is used to save the color of the object [23]. This color will only be applied when drawing a filled object. Figure 3.11 shows a filled object on the right, having the filling been carried out as in Listing 3.17.

Listing 3.17: Object filled with color.

```
1 <script type="text/javascript">
2    quads.GLY_FILL_COLOR(255,0,0,1);
3 </script>
```

With this method, all faces of the object are filled with the same color.

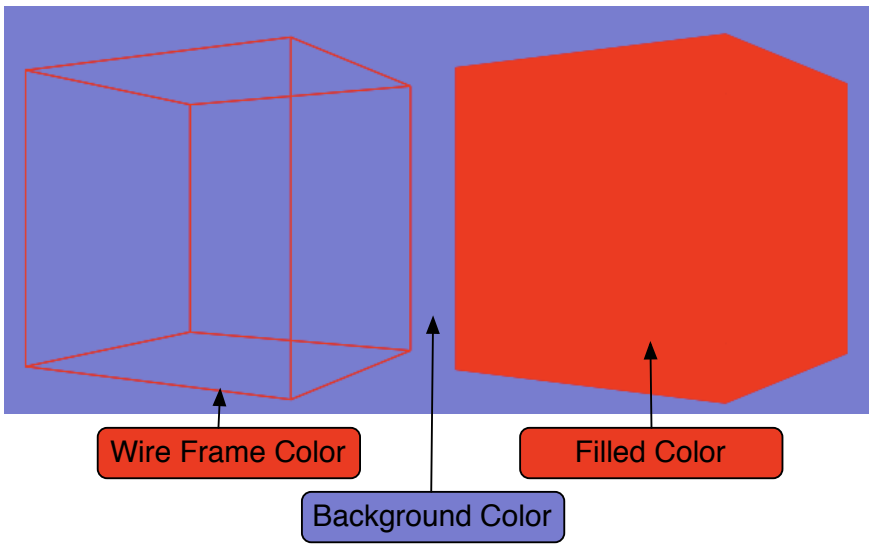


Figure 3.11: Illustration of applying the background, wireframe, and fill coloring methods. On the left, a red wireframe cube, on the right, a red filled cube.

### 3.9.4 Flat Shading

Until know, the only way to assign a color to an object was by filling all object faces with the same color. However, this filling process is not realistic in 3D. A more realistic shading technique is given by the flat shading algorithm. For that, it shades each face of the object taking into account the light position-defined vector and each face normal. If these vectors form an angle of $\theta = 180$ degrees (point to each other), the surface has maximum color intensity and becomes brighter, but when this angle decreases to zero, the color intensity also decreases and becomes darker.

The flat shading algorithm has thus the following steps:

- *To calculate the face normals.* This step is identical to the first step of the back-face culling.
- *To compute the color intensity of each face.* This is done by computing the dot product of two vectors: face normal and light position-defined vector. This product provides us with the color intensity.
- *To compute the color of each face.* Calculated the color intensity $I$, between 0 and 1, of each face in the previous step, it remains to compute its color by using Equations (3.15):

$$R' = R.I \tag{3.15a}$$
$$G' = G.I \tag{3.15b}$$
$$B' = B.I \tag{3.15c}$$

Where $R$, $G$, and $B$ values are the color parameters introduced by the user, and $R'$, $G'$, and $B'$ values compound to the final color of each face. Now, we have different RGB colors for each face of the object. Notice that the color is constant for the same face, but different faces have different intensities, i.e. it is brighter or darker due to the intensities values. The Glypher3D flat shading method is specified as follows:

- GLY_FLAT_SHADING(int R, int G, int B, float A).

Like the wireframe and fill color methods, it is possible the change the transparency of the object. Again, the flat shading method uses the canvas context `fillStyle` method to save the color of each face of the object. This color is only applied to the faces of the object when the object is drawn. As an example, let us apply the flat shading algorithm to a triangle as in Listing 3.18.

Listing 3.18: Flat shading an object.

```
1 <script type="text/javascript">
2   triangle.GLY_FLAT_SHADING(255,0,0,1);
3 </script>
```

Despite of being more realistic than the fill color method, the flat shading algorithm still lacks realism in shading objects of a scene (see Figure 3.12), because all points of the same face have the same color. However, the flat shading algorithm does not calculate the intensities for each pixel, but only for each face, which makes the rendering of an object a faster process than the one with the Gouraud shading.

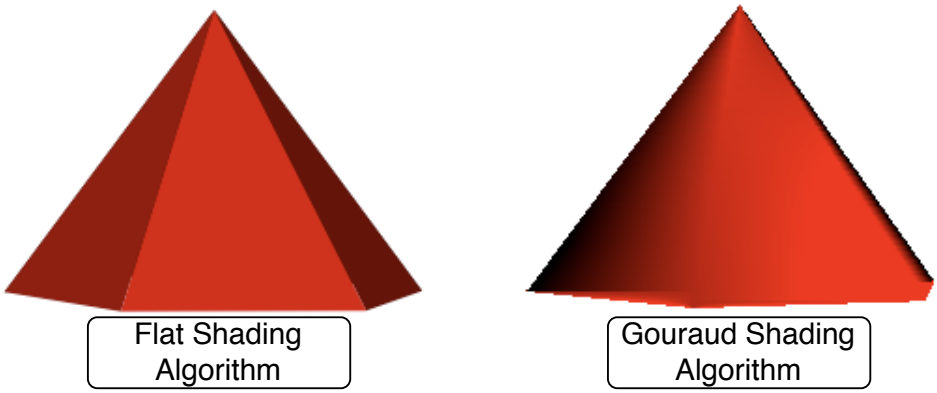


Figure 3.12: Glypher3D flat shading and Gouraud shading algorithms. On the left, a flat shaded pyramid, on the right, a Gouraud shaded pyramid.

### 3.9.5 Gouraud Shading

In order to try to achieve an even more realistic 3D scene, we have implemented the Gouraud shading algorithm. Unlike flat shading, this algorithm does not compute a color per face, but a color per pixel associated to such a face. To calculate the intensities for each pixel of each face of the object, and ultimately the pixels colors, a sequence of steps have to be followed.

#### 3.9.5.1 Vertex Normals

The first step is the calculation of the vertex normals. This process consists in averaging the values of the face normals that share the same vertex. For example, if one vertex is shared by four faces, the vertex normal is the average of these four face normals. We assume that the Gouraud shading algorithm applies to triangular faces, so any object compounded from other polygons requires its splitting into triangles.

#### 3.9.5.2 Color Intensity of Vertices

Then, using the calculated vertex normals, it was implemented a Phong reflection model [22] to compute the vertices intensities through a combination of diffuse reflection, ambient light and specular reflection intensities.

Objects do not emit light, but they reflect any light that illuminates them. The diffuse light or diffuse reflection depends on the light and object positions. To calculate the diffuse reflection intensity $I_d$, first, it is calculated the dot product between each vertex normals $V_N$ and the light vector $L$. Then, this value is multiplied by a $K_d$ constant between 0 and 1, as it is done in Equation (3.16). This constant is an parameter of the Gouraud shading algorithm and represents an approximated value of the diffuse reflectivity.

$$I_d = K_d(L.V_N) \tag{3.16}$$

The ambient light does not have direction or position and is a result of multiple reflections. It illuminates a face from all directions. The intensity of the ambient light $I_{ambient}$, is calculated using Equation (3.17).

$$I_{ambient} = I_a.K_a \tag{3.17}$$

In Equation (3.17), $I_a$ is the intensity of the ambient light and $K_a$ is the ambient refection coefficient. $I_a$ and $K_a$ are two constants with values between 0 and 1, also passed as parameters to the Gouraud shading algorithm.

The specular light or specular reflection is intended to create glossy regions on the surface of the object. It takes into account the light position, the viewer position and the object position. The calculation of the specular intensity involves two steps: first, it is calculated the $S$ term, which is an approximation of the reflection vector, using Equations (3.18):

$$S_x = (L_x + V_x)/2 \tag{3.18a}$$
$$S_y = (L_y + V_y)/2 \tag{3.18b}$$
$$S_z = (L_z + V_z)/2 \tag{3.18c}$$

Where $L = (L_x, L_y, L_z)$ and $V = (V_x, V_y, V_z)$ are the light vector and the viewer vector respectively.

Because the calculation of the reflection vector requires more computational work, instead of this vector, it is calculated the $S$ term, which basically is a unit normal with a direction between the light vector $L$ and the viewer vector $V$.

Finally, to obtain the specular intensity $I_s$, it is first calculated the dot product between the vertices normals vector $V_N$ and the $S$ term. Then, this value is multiplied by a $K_s$ constant between 0 and 1, as is done in Equation (3.19). The $K_s$ constant is also an parameter of the Gouraud shading algorithm and denotes the specular reflection coefficient.

$$I_s = K_s(S.V_N) \tag{3.19}$$

By combining the diffuse reflection, the ambient light and the specular reflection, we obtain Equation (3.20) for the calculation of the intensities $I_{vertex}$ for each vertex.

$$I_{vertex} = I_{ambient} + L_i.(I_d + I_s^n) \tag{3.20}$$

In Equation (3.20), $I_{ambient}$ is the intensity of the ambient light, $I_d$ is the intensity of the diffused light, $I_s$ is the intensity of the specular light, $L_i$ is the intensity of the light source and also a constant with values between 0 and 1 (passed as an parameter to the Gouraud shading function), and finally, the constant $n$ represents the positive value of the glossiness of the surface. Smaller the glossiness value, bigger the highlight on the surface.

#### 3.9.5.3 Color Intensity of Pixels

After the calculation of the color intensities of vertices, the next step is to calculate the color intensity for each pixel of each face boundary (i.e., edges) of the object. Then, we calculate the intensities for each pixel of each face interior along scan lines using linear interpolation. Figure 3.13 illustrates this process for a triangle.

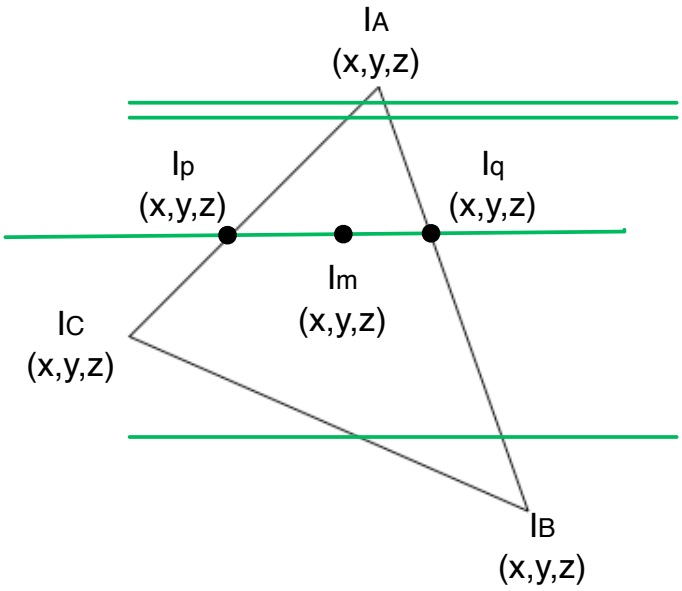


Figure 3.13: Intensities for each pixel of the triangle along scan lines using linear interpolation.

The generic equations to calculate the intensities of the pixels of each edge of the object are Equation (3.21), for the pixels on the edge $\bar{AC}$, and Equation (3.22), for the pixels on the edge $\bar{AB}$ [22].

$$I_p = (1/A_y - C_y).(I_A.(p_y - C_y) + I_C.(A_y - p_y)) \tag{3.21}$$

In Equation (3.21), $I_p$ corresponds to the pixels intensities on the left edge, $A_y$ corresponds to the first vertex $y$ position, $C_y$ corresponds to the third vertex $y$ position, $I_A$

corresponds to the intensity on the first vertex, $p_y$ corresponds to the $y$ positions of the left pixels, and $I_C$ corresponds to the intensity on the third vertex.

$$I_q = (1/A_y - B_y).(I_A.(q_y - B_y) + I_B.(A_y - q_y)) \tag{3.22}$$

In Equation (3.22), $I_q$ corresponds to the pixels intensities on the right edge, $A_y$ corresponds to the first vertex $y$ position, $B_y$ corresponds to the second vertex $y$ position, $I_A$ corresponds to the intensity on the first vertex, $q_y$ corresponds to the $y$ positions of the right pixels, and $I_B$ corresponds to the intensity on the second vertex.

To obtain just the positions of the pixels from the edges was used the Bresenham line algorithm. To calculate the intensities for each pixel of each face interior along scan lines using linear interpolation, we use Equation (3.23).

$$I_m = (1/q_x - p_x).(I_p.(q_x - m_x) + I_q.(m_x - p_x)) \tag{3.23}$$

In Equation (3.23), $I_m$ corresponds to the pixels intensities in the interior of a face, $q_x$ corresponds to the $x$ positions of the right pixels, $p_x$ corresponds to the $x$ positions of the left pixels, $I_p$ corresponds to the pixel intensities on the left edge, $m_x$ corresponds to the $x$ positions of the pixels in the interior of a face, and $I_q$ corresponds to the pixel intensities on the right edge.

To obtain the positions of the pixels in the interior of a face ($m_x$), were covered all the pixels between the pixels on the left edge ($p_x$) and the pixels on the right edge ($q_x$).

Now that all the pixels intensities are calculated, the last step on the Gouraud shading algorithm is to set the pixels colors. To do that, we use Equations (3.24):

$$R' = R.I_{pixel} \tag{3.24a}$$
$$G' = G.I_{pixel} \tag{3.24b}$$
$$B' = B.I_{pixel} \tag{3.24c}$$

The $R$, $G$, and $B$ values are the color parameters introduced by the user, and the $R'$, $G'$, and $B'$ values correspond to the colors intensities of pixels.

Then, the Glypher3D Gouraud shading algorithm uses the following HTML5 canvas element, pixel-based manipulation methods, to save the pixels colors (only applied to pixels when drawing):

- `imagedata = canvas context.createImageData(int width, int height)`. This method returns an imagedata object with the given dimensions. The imagedata object can access to the pixels methods to change their size or color. The width and height attributes correspond to the initial pixel size [23]. This method is supported by the Firefox, Safari and Chrome browsers.

- `imagedata = canvas context.getImageData(int x, int y, int width, int height)`. This method also returns an imagedata object, however, this object can access to the pixel data from the given rectangle position of the canvas [23]. This method is only supported by the Opera browser.

Both methods provides an `imagedata` object to manipulate the pixel data inside the canvas area. However, because not all canvas-compatible browsers support the `createImageData` method, was also implemented the `getImageData` method. Through the `imagedata` object is possible to access to each one of the object pixels positions and set their new RGB colors.

The Glypher3D Gouraud shading method is the following:

- GLY_GOURAUD_SHADING(float $I_a$, float $K_a$, float $K_d$, float $L_i$, float $K_s$, int $n$, int R, int G, int B, int A).

Similar to the previous shading methods, Gouraud shading allows for changing the transparency of the object. The difference is that now the alpha value is between 0 and 255, where the maximum value (255) corresponds to an object without transparency (the same as 1 on the previous shading methods).

As an example, let us apply the Gouraud shading to a triangle fan, as illustrated in Listing 3.19 and Figure 3.12.

Listing 3.19: Gouraud shading an object.

```
1 <script type="text/javascript">
2     triangle.GLY_GOURAUD_SHADING(0.2,0.2,1,1,1,0,255,0,0,255);
3 </script>
```

In Glypher3D, the Gouraud shading algorithm is the only method that directly handles pixels of the HTML5 canvas element. Because of all the mathematical calculations are performed in JavaScript, on the client-side, Gouraud shading is slower on browser than OpenGL on desktop applications that use the capabilities of graphics cards. However, as expected, the Gouraud technique provides a better and smoother look of the object, as a color gradient exists on the surface.

## 3.10 Projection Transformations

A scene and the objects therein are defined in the 3D space. But, in order to display this scene and its objects onto a 2D rectangular canvas area of the browser, the scene must be projected onto a plane in 3D, which is then mapped onto the 2D context of canvas. Therefore, before displaying a scene, we must apply projection transformations to map 3D coordinates onto 2D coordinates. There are two main projection techniques: parallel projection and perspective projection [16, 29].

### 3.10.1 Parallel Projection

The parallel projection (front orthographic projection), is a less realistic projection, because it ignores the $z$ coordinate or depth coordinate, i.e. all the points are projected along parallel lines (i.e. the visual rays) because we assume that the viewer is located at infinity. Therefore,

there is no foreshortening, i.e. if we had two objects within the scene, one closer and another further away, they would be rendered with the same size [16, 29]. Glypher3D uses by default the parallel projection, because the rendering process only involves the $x$ and $y$ coordinates. The $z$ coordinate is simply discarded.

### 3.10.2 Perspective Projection

The perspective projection provides a more realistic visualization (similar to the human visual system) of the object and does not ignore the $z$ coordinate, as it happens in parallel projection. This means that the more distant an object is from the center of projection (origin), smaller it appears, i.e. there is foreshortening. Unlike the parallel projection, in which the parallel lines only meet at infinity, in perspective projection, parallel lines meet at the viewer position who is located to a finite distance in the positive $z$-axis [16, 29]. In these circumstances, the perspective projection is given by the equation system (3.25).

$$\begin{cases} x' = x/z \\ y' = y/z \\ z' = z \end{cases} \tag{3.25}$$

The higher the value of $z$, smaller the values of $x'$ and $y'$, and the final point becomes closer to the viewer. Equation system (3.25) present a problem because the $z$ value cannot be zero (division by zero). To solve this problem, we introduced two new variables: the $width$ and $height$ of the canvas area, changing the equation system (3.25), to the equation system (3.26).

$$\begin{cases} x' = x/(width - z) \\ y' = y/(height - z) \\ z' = z \end{cases} \tag{3.26}$$

The $width$ and $height$ variables represent the dimensions of the canvas area (in pixels), and are used to maintain the object within it. However, all points still converge to the viewer, and the object continues very small. Therefore, to put the object on a visible scale and with perspective, we multiply the $x$ and $y$ values, respectively, by the $width$ and $height$ values, as shown in equation system (3.27).

$$\begin{cases} x' = (width.x)/(width - z) \\ y' = (height.y)/(height - z) \\ z' = z \end{cases} \tag{3.27}$$

This method of perspective projection bypasses the normalization process of coordinates as a way to speed up the rendering process. The perspective projection method is the following:

- GLY_PERSPECTIVE_PROJECTION().

An example of its use is provided in Listing 3.20.

Listing 3.20: Perspective projection applied to a triangle.

```
1 <script type="text/javascript">
2     triangle.GLY_PERSPECTIVE_PROJECTION();
3 </script>
```

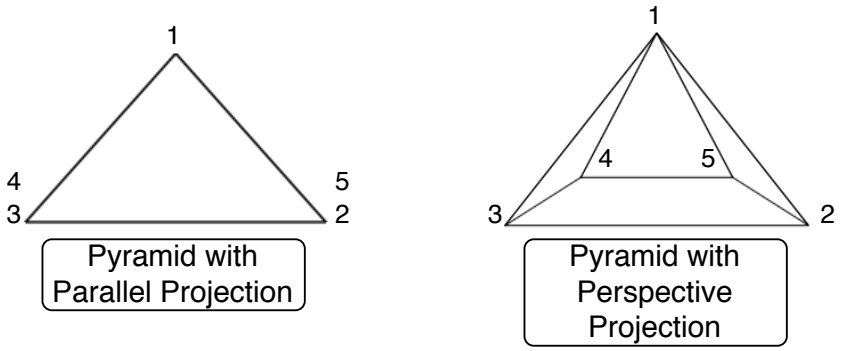


Figure 3.14: Glypher3D parallel and perspective projections applied to a pyramid.

Figure 3.14 shows a pyramid displayed with the parallel and perspective projections.

## 3.11 Display

The final stage of the Glypher3D rendering pipeline is to display all objects that compound a 3D scene on browser. This stage allows us to output the scene and its objects onto the 2D canvas rendering context. The objects of the scene can be drawn filled (with fill color or shading algorithms) or wireframe (with the wireframe color method). To actually draw on the 2D canvas rendering context, Glypher3D drawing methods take advantage of some of the HTML5 canvas element, drawing and pixel-based manipulation methods [23].

For the canvas element, the basic drawing primitive is the rectangle. However, the Glypher3D drawing methods, that allow us to draw lines, triangular and quadrangular shapes, uses the complex shapes or paths methods of the canvas element. A path is compounded by a list of points (vertices of objects) connected by lines (edges of objects), where is possible to color just the lines, i.e. wire-framing the objects, or the interior of the path, i.e. filling or flat shading the objects [23].

The Glypher3D drawing methods make usage of the following drawing path methods of the canvas element in order to draw lines, triangular and quadrangular shapes:

- `canvas context.beginPath()`. It allows to begin the drawing path of the object.

- `canvas context.moveTo(x,y)`. It specifies the starting point of the path, i.e. it starts drawing from this point.
- `canvas context.lineTo(x,y)`. It draws a line from the previous point to the current point. It draws the edges of 3D objects.
- `canvas context.closePath()`. It closes the path of the object by drawing a line from the current point to the starting point. It forms a closed path.
- `canvas context.stroke()`. It colors the lines or edges of the object with the `strokeStyle` attribute defined with the Glypher3D wireframe color method.
- `canvas context.fill()`. It fills the object with the `fillStyle` attribute defined with the Glypher3D fill color or flat shading methods.

These drawing path methods are not used in Gouraud shading. Instead, we use the following pixel-based manipulation drawing method:

- `canvas context.putImageData(image data object, int x, int y)`. This method uses the `imagedata` object (that contains all information of the pixels, including their size and colors) to draw the pixels at specific positions [23].

Also, through this pixel-based manipulation method, we can draw just the vertices of the object through the following Glypher3D method:

- GLY_DRAW_VERTEXS(int size, int R, int G, int B).

The previous method is used as in Listing 3.21.

Listing 3.21: Drawing the vertices of an object.

```
1 <script type="text/javascript">
2   quads.GLY_DRAW_VERTEXS(5,0,250,0);
3 </script>
```

As mentioned before, Glypher3D allows for drawing filled or wireframe objects. To draw a specific object, we use the following method:

- GLY_DRAW(string).

If this method is used without parameters, the object will be drawn with color or with shading. However, if we pass the identifier "GLY_WIREFRAMED" as argument, the object will be drawn as a wireframe object.

Therefore, the drawing of an specific object with color or shading is done as in Listing 3.22. But drawing a wireframe object is done as in Listing 3.23. Figure 3.15 shows a scene with two objects. The first is a wireframe object, while the second is a filled object. The vertices of both objects appear drawn with square marks.

Listing 3.22: Drawing a filled or shaded object.

```
1 <script type="text/javascript">
2   quads.GLY_DRAW();
3 </script>
```

Listing 3.23: Drawing a wireframe object.

```
1 <script type="text/javascript">
2   quads.GLY_DRAW("GLY_WIREFRAMED");
3 </script>
```

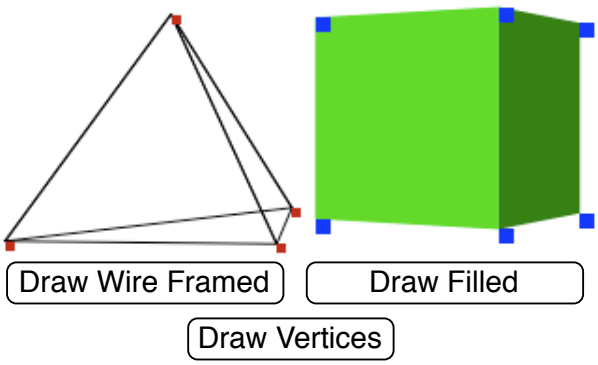


Figure 3.15: Glypher3D drawing methods. On the left, a wireframe tetrahedron, on the right, a shaded cube. For both, all visible vertices are drawn.

When rendering the scene, if the object has suffered any modeling transformation (e.g. a rotation), the final position of the object changes. So, in order to redraw this object at the new position it is needed to clear the rendering area first. This must be done through the following Glypher3D method:

- GLY_CLEAR_CANVAS().

This method takes advantage of the following HTML5 canvas element method:

- `canvas context.clearRect(int x, int y, int width, int height).`

This method clears all pixels of the canvas area defined by the width and height parameters. The $x$ and $y$ are the starting point of the canvas rectangular area [23].

## 3.12 Summary

To conclude this chapter, it can be said that it is possible to create a 3D graphics pipeline using only graphics algorithms encoded in JavaScript, together with the HTML5 canvas element. This Glypher3D rendering pipeline, although simpler than a classic rendering pipeline, provides the essential facilities to create 3D scenes directly on browser. It allows us to create a viewport, 3D scenes with graphics primitives (i.e. with points, lines, line strips, line loops, triangles, triangle strips, triangle fans, quad and quad strips), apply geometric transformations (i.e. rotation, translation, scaling and shearing), define the light position, apply hidden surface removal algorithms (i.e. back-face culling and painter's algorithm), apply shading (i.e. flat shading and Gouraud shading), projections transformations (i.e. perspective projection) and finally to display the 3D scene on the 2D viewport.

# Chapter 4

# Results

This chapter exposes some of the results that can be obtained with Glypher3D. This includes a comparison of the library rendering performance (using an flat and *Gouraud shaded* object) on different platforms and multiple browsers. It is also compared the rendering time of a 2D object, using the canvas element and SVG.

## 4.1 Glypher3D Scenes

Some of the results obtained using the Glypher3D graphic library were already shown during Chapter 3. For example, Figures 3.11 and 3.12. Here, we show some self-contained examples of what can be created using this library.

Listing 4.1 shows how to create a full Glypher3D scene, compounded by an triangle fan shape (with 7 vertices and 6 faces) that continuously rotates on the $y$-axis, with the painter's algorithm. The scene is rendered using Gouraud shading and perspective projection.

Listing 4.1: Full Glypher3D scene.

```
1  <!DOCTYPE html>
2  <html>
3  <head>
4    <meta http-equiv="Content-Type" content="text/html;
5  ␣␣charset=UTF-8" />
6    <title>Glypher3D Gouraud Fan</title>
7    <script type="text/javascript"
8    src="glypher3dlib/Glypher3D.js">
9    </script>
10   <script type="text/javascript">
11     var angle = 0;
12     function init(){
13       var canvas = document.getElementById('area');
14       GLY_INIT_2D(canvas);
15       GLY_CLEAR_CANVAS();
```

```
16
17          angle+=0.025;
18
19          var g1 = new GLY_POINT_3D(0,150,0);
20          var g2 = new GLY_POINT_3D(50,-100,100);
21          var g3 = new GLY_POINT_3D(-50,-100,100);
22
23          var g4 = new GLY_POINT_3D(-100,-100,0);
24          var g5 = new GLY_POINT_3D(-50,-100,-100);
25          var g6 = new GLY_POINT_3D(50,-100,-100);
26          var g7 = new GLY_POINT_3D(100,-100,0);
27
28          var vertices = new Array(g1,g2,g3,g4,g5,g6,g7,g2);
29          var fan = new GLY_SHAPE(vertices,"GLY_TRIANGLE_FAN");
30
31          //fan.GLY_POINTS_TOSTRING();
32          fan.GLY_ROTATION_Y(angle);
33          //fan.GLY_TRANSLATION(-100,0,0);
34          //fan.GLY_SCALING(0.5,0.8,1);
35          //fan.GLY_SHEARING(0.8,0,0);
36
37          GLY_LIGHT_POSITION(0,0,300);
38
39          fan.GLY_PAINTERS_ALGORITHM();
40          //fan.GLY_BACKFACE_CULLING();
41
42          //fan.GLY_WIREFRAME_COLOR(3,253,25,1);
43          //fan.GLY_FILL_COLOR(3,253,25,1);
44          //fan.GLY_FLAT_SHADING(3,253,25,1); //R,G,B,A
45          fan.GLY_GOURAUD_SHADING(0.2,0.2,1,1,1,0,3,253,25,255);
46
47          fan.GLY_PERSPECTIVE_PROJECTION();
48
49          fan.GLY_DRAW(); //"GLY_WIREFRAMED"
50          //fan.GLY_DRAW_VERTEXS(8,0,0,255);
51
52          //GLY_BACKGROUND_COLOR(canvas,100,100,100);
53      }
54      function Rotate(){
55          setInterval(init,60);
56          //init();
57      }
58          </script>
59  </head>
```

```
60 <body onload="Rotate();">
61   <canvas id="area" width="600" height="600"></canvas>
62 </body>
63 </html>
```

The graphics output of the scene encoded in Listing 4.1 is shown in Figure 4.1. Notice that by using the Gouraud shading, the object has a smoother and more realistic look.

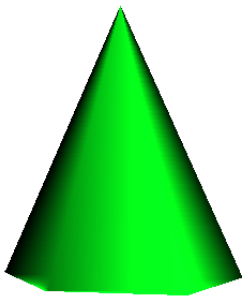

Figure 4.1: Glypher3D triangle fan shape with rotation, painter's algorithm, Gouraud shading, and perspective projection facilities. Scene rendered with Safari 4 on Mac OS X v.10.5.7.

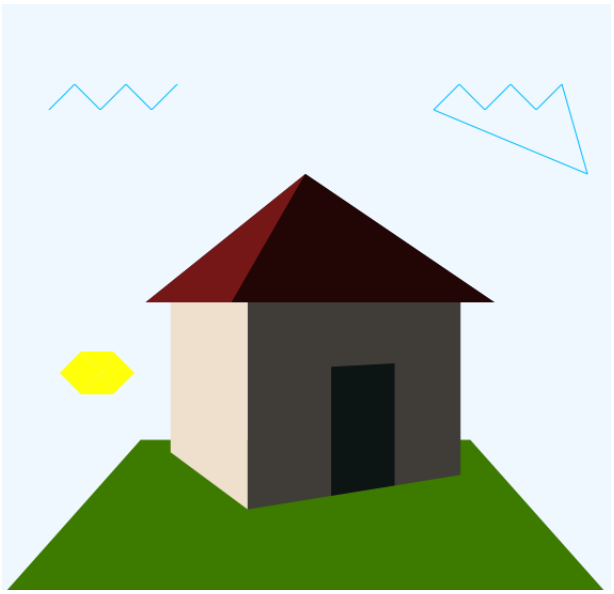

Figure 4.2: Glypher3D scene with multiple shapes, i.e, compounded of lines, triangles and quad primitives. Scene rendered with Safari 4 on Mac OS X v.10.5.7.

Figure 4.2 shows a more complex Glypher3D scene, i.e. with multiple objects. In that scene we use various Glypher3D graphics primitives. For the house, floor, and door we use quad primitives. The roof and sun are modeled using triangle fan primitives. Finally, the clouds are build up using two lines primitives: a line strip on the left and a line loop on the right.

An example of a Glypher3D web application is shown in Figure 4.3. In this application it is possible to see a fully functional web site with a Glypher3D scene, in which we can interact with the object through various interaction buttons.

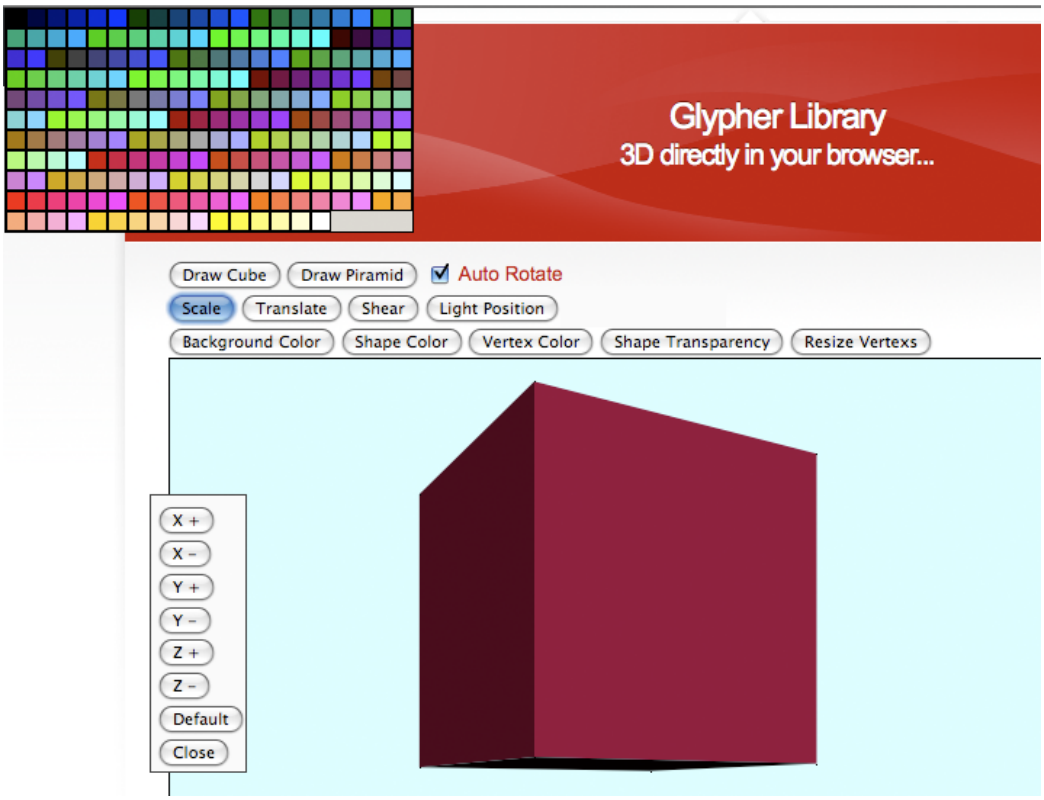


Figure 4.3: Fully functional Glypher3D web application running with Firefox 3.5 on Mac OS X v.10.5.7.

## 4.2 Rendering Performance

Let us now compare how the Glypher3D library performs on different platforms (Windows and Mac), and how it performs on the most common, canvas-compatible browsers, like Firefox 3.5, Safari 4, Opera 10 Beta and Chrome 2, by comparing the rendering performance between an *flat shaded* object and an *Gouraud shaded* object. It is also tested the Glypher3D library on Internet Explorer 7 with the ExplorerCanvas plug-in release 3, and the rendering

performance of an 2D object, using the HTML5 canvas element and SVG 2D graphics technologies.

To test the rendering performance of Glypher3D on the mentioned platforms and browsers, we have used multiple instances of the triangle fan shape encoded in Listing 4.1 together with two shading algorithms: Gouraud shading and flat shading.

The rendering time of an Glypher3D scene was obtained by subtracting the time between the beginning of the scene, i.e, the creation of the rendering context, and the end of the scene (output to the screen).

### 4.2.1 Mac Platform

The rendering performance of Glypher3D, from one to ten *Gouraud shaded* triangle fan shapes, in a MacBook 2.2 Ghz Intel Core 2 Duo, with Mac OS X v10.5.7, and using the Firefox 3.5, Safari 4 and Opera 10 browsers, is shown in Figure 4.4.

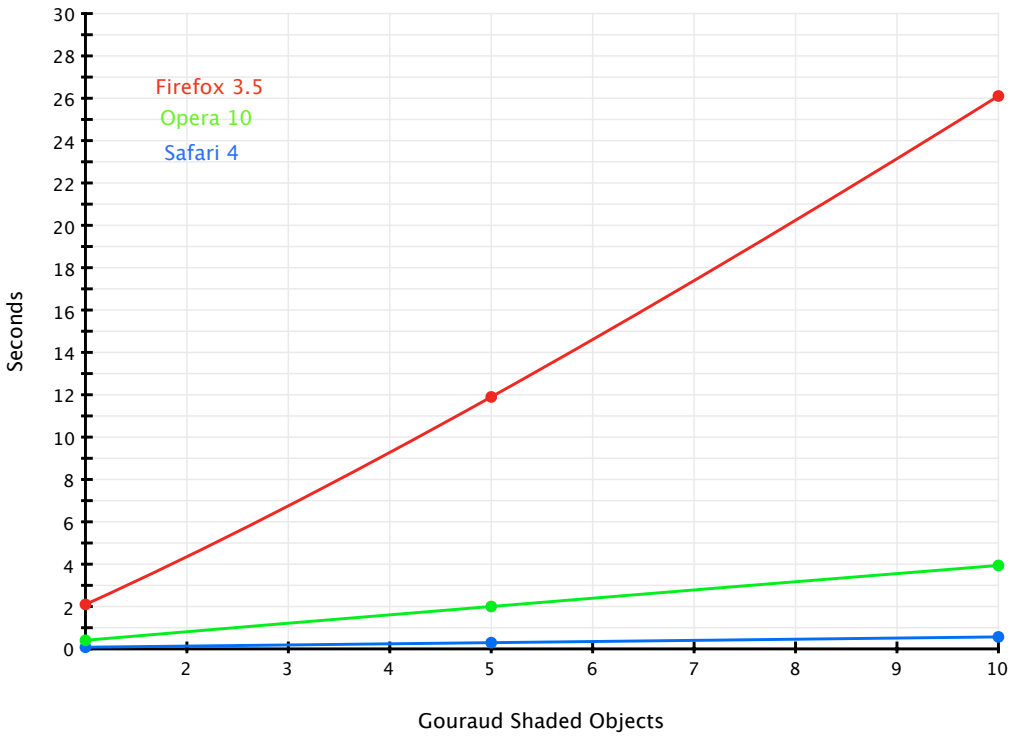


Figure 4.4: Glypher3D rendering performance of *Gouraud shaded* objects with Firefox 3.5, Safari 4 and Opera 10 on Mac OS X v.10.5.7.

Analyzing the chart of Figure 4.4, it can be said that the rendering performance of an *Gouraud shaded* object is much faster when using the Safari 4 browser. This is due to the fact that this browser has better JavaScript performance than the other browsers. With Safari 4, the rendering time of an Glypher3D scene, from one to ten *Gouraud shaded* objects, is always smaller than one second. Opera 10 has the second best performance, and renders an

Glypher3D scene with ten *Gouraud shaded* objects in approximately four seconds. On the other hand, Firefox 3.5 has the worst performance, especially when rendering five or more *Gouraud shaded* objects.

The rendering performance of Glypher3D, from one to ten *flat shaded* triangle fan shapes, in a MacBook 2.2 Ghz Intel Core 2 Duo, with Mac OS X v10.5.7, and using the Firefox 3.5, Safari 4 and Opera 10 browsers, is shown in Figure 4.5.

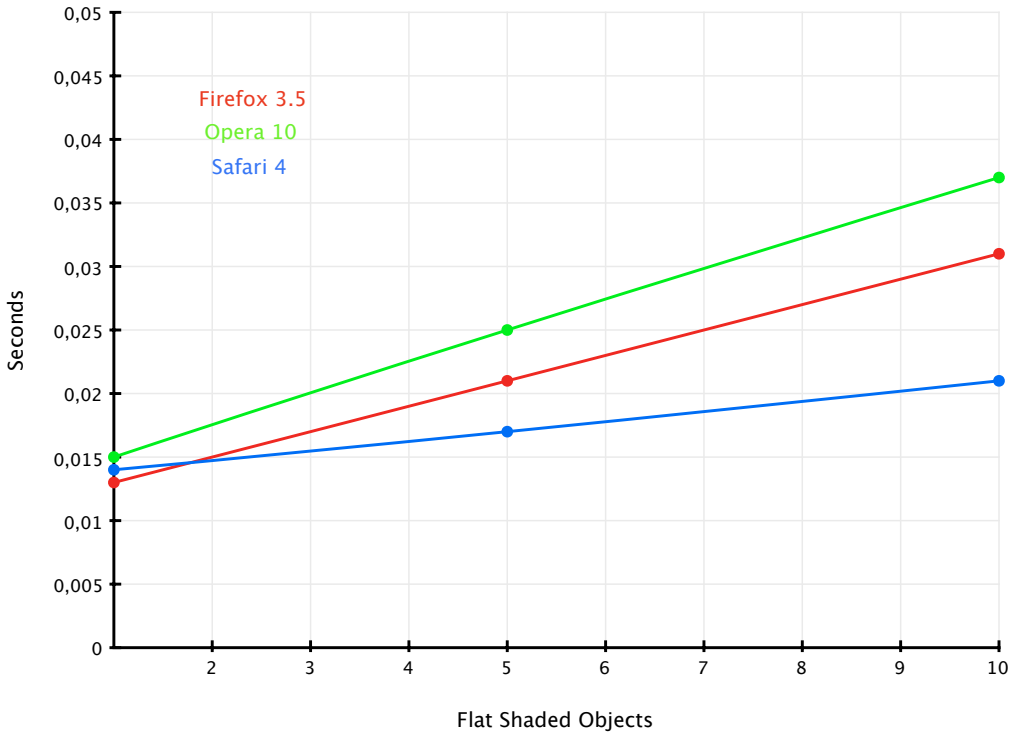


Figure 4.5: Glypher3D rendering performance of *flat shaded* objects with Firefox 3.5, Safari 4 and Opera 10 on Mac OS X v.10.5.7.

The chart on Figure4.5 shows that rendering *flat shaded* objects is much faster than rendering *Gouraud shaded* objects (Figure 4.4). This happens because the flat shading algorithm does not deal with pixel-based calculations, like it happens with the Gouraud shading algorithm. This performance test shows that the Safari 4 browser has the better performance, and the Firefox 3.5 slightly performs better than the Opera 10 browser. The rendering times of an Glypher3D scene, even with ten *flat shaded* objects, are very fast (between 0.020 and 0.040 seconds).

### 4.2.2 Windows Platform

The rendering times of Glypher3D, from one to ten *Gouraud shaded* triangle fan shapes, in a P4 3.0 Ghz, with Windows XP SP3, and using the Firefox 3.5, Safari 4, Opera 10 and Chrome 2, are illustrated in Figure 4.6.

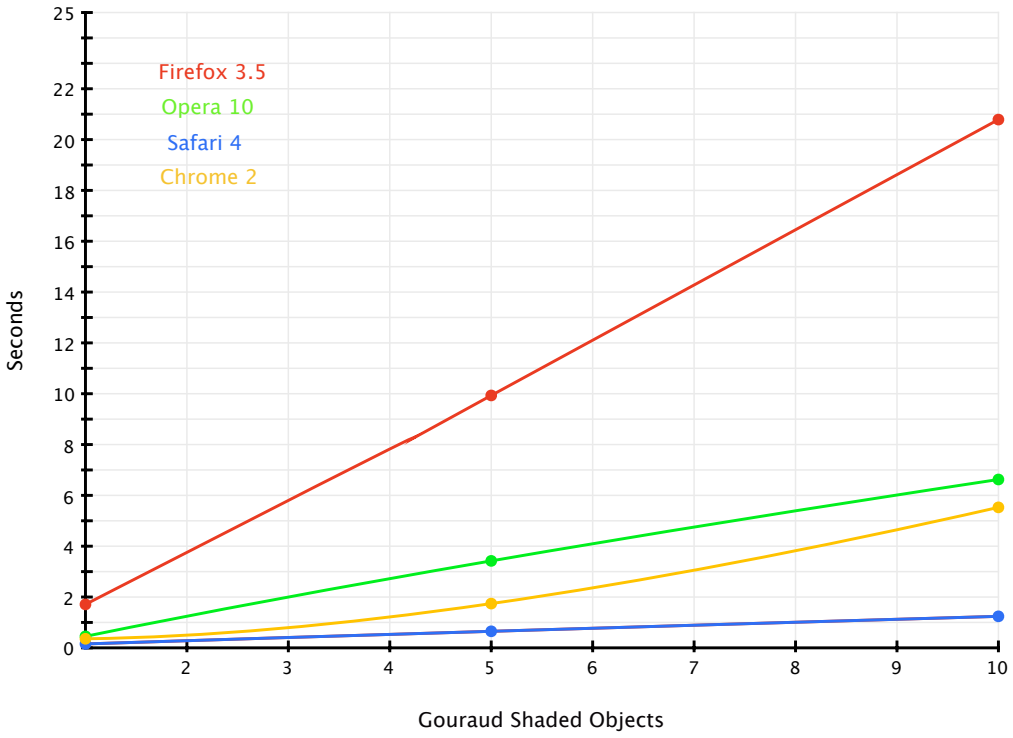


Figure 4.6: Glypher3D rendering performance of *Gouraud shaded* objects with Firefox 3.5, Safari 4, Opera 10 and Chrome 2 on Windows XP SP3 .

Using the Windows platform, it was possible to test the performance of the Glypher3D library on the Google's Chrome 2 browser. The chart of Figure 4.6 shows that the rendering performance of Glypher3D is only slightly lower (due to an worst processor performance) to the one obtained in the Mac platform (Figure 4.4). However, the Safari 4 browser continues to have the better results than the others on Windows XP. The Chrome 2 browser has a slighter better performance than the Opera 10 browser, and the Firefox 3.5 holds the last place when rendering Glypher3D scenes with *Gouraud shaded* objects.

Finally, the rendering performance of Glypher3D, from one to ten *flat shaded* triangle fan shapes, in a P4 3.0 Ghz, with Windows XP SP3, and using the Firefox 3.5, Safari 4, Opera 10, Chrome 2 and Internet Explorer 7, is illustrated in Figure 4.7.

The performance results showed in the chart of Figure 4.7 demonstrates that the flat shading algorithm performs faster than the Gouraud shading algorithm (already seen in Figure 4.5). Just like in the Mac Platform, on Windows, the Safari 4 browser has the better performance. The Chrome 2 browser has the second best performance, the Firefox 3.5 performs slightly better than the Opera 10 browser, and the Internet Explorer 7 has the worst performance. While with Safari 4, Chrome 2, Firefox 3.5, and Opera 10 browsers, the rendering time of an Glypher3D scene, with ten *flat shaded* objects, is around the $0.050$ seconds, with Internet Explorer 7 is around $0.3$ seconds.

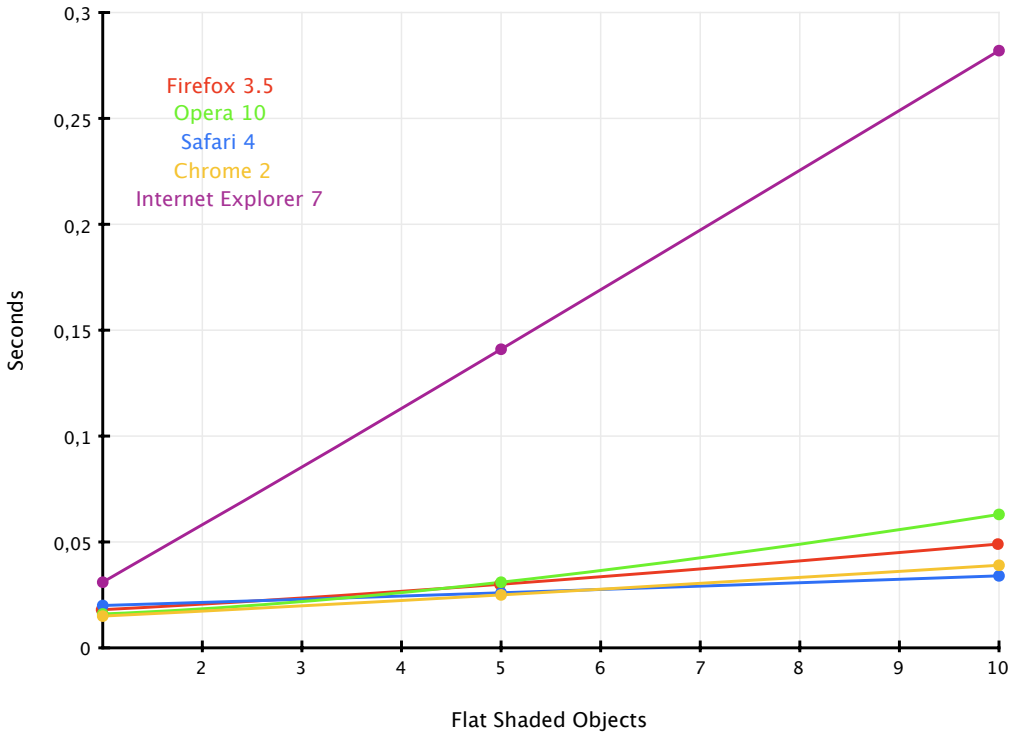


Figure 4.7: Glypher3D rendering performance of *flat shaded* objects with Firefox 3.5, Safari 4, Opera 10, Chrome 2 and Internet Explorer 7 on Windows XP SP3.

Using the Windows platform, it was possible to test the library on Internet Explorer 7, through the Google's ExplorerCanvas plug-in release 3. However, this plug-in only adds to this browser the support for the simple drawing functions of the canvas element, like those used by the flat shading algorithm, therefore, it does not support the pixel-based manipulation canvas functions. Consequently, it does not support the Gouraud shading algorithm. It has a slower rendering process than the other browsers because it needs to translate the canvas functions to VML.

Comparing the results obtained with the multiple browsers, and with the shading algorithms available in the Glypher3D library, we can conclude that the Glypher3D library has a better performance on the Safari 4 browser. Also, as expected, rendering *flat shaded* objects is faster than rendering *Gouraud shaded* objects.

### 4.2.3 HTML5 and SVG

Let us now compare the performance of HTML5 to SVG. For this purpose, it was encoded a simple 2D square (100 pixels width by 100 pixels height), using graphic primitives of respective technologies. The rendering times of these two technologies using 1, 50, 100, 500 and 1000 squares, in a MacBook 2.2 Ghz Intel Core 2 Duo, with Mac OS X v10.5.7, and using the Firefox 3.5 browser, are shown in Figure 4.8.

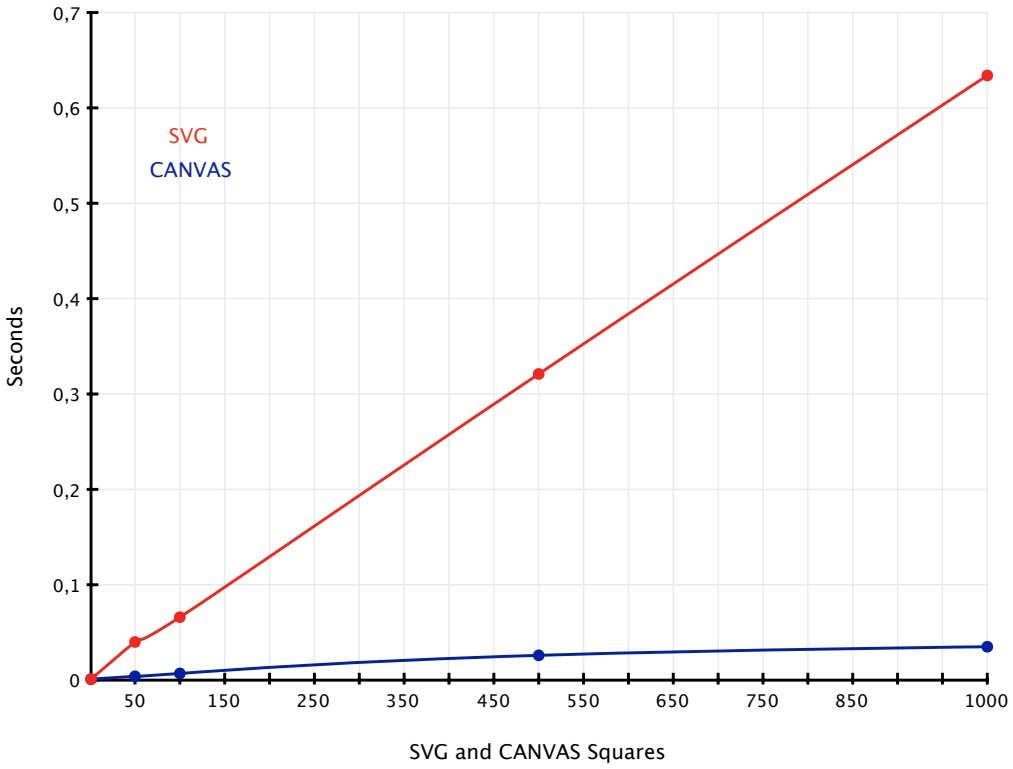


Figure 4.8: Rendering performance between the HTML5 canvas element and SVG with Firefox 3.5 on Mac OS X v.10.5.7.

The rendering times of these two technologies using 1, 50, 100, 500 and 1000 squares, in a P4 3.0 Ghz, with Windows XP SP3, and using the Firefox 3.5 browser, are shown in Figure 4.9.

Analyzing the charts of Figures 4.8 and 4.9, it can be said that the rendering performance of the HTML5 canvas element, when rendering a substantially large number of objects, is better than the rendering performance of SVG on both platforms. This was an important test to determine the best 2D technology to use in the Glypher3D library.

## 4.3 Summary

As shown in this chapter, Glypher3D seems to be a good technology to render 3D contents on a web page, because it allows us to integrate 3D logos, models, advertisements, etc, in a web page. Besides, we show that the Glypher3D library has a better performance on the Safari 4 browser. As expected, rendering *flat shaded* objects is faster than rendering *Gouraud shaded* objects. This happens because the Gouraud shading algorithm uses the HTML5 canvas element pixel-based manipulation methods, which are more computationally intensive and time-consuming. We also demonstrate that Glypher3D is a multi-platform and cross

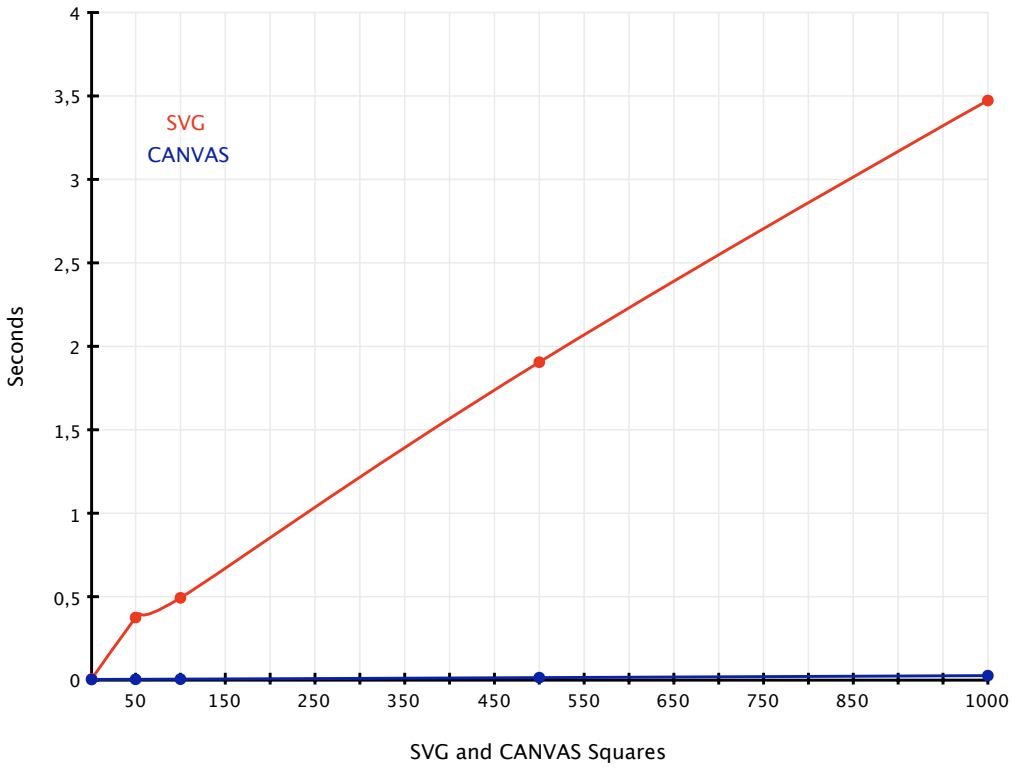


Figure 4.9: Rendering performance between the HTML5 canvas element and SVG with Firefox 3.5 on Windows XP SP3.

browser library. However, older browsers and the Internet Explorer cannot take advantage of all the Glypher3D functionalities due to their lack of support for the HTML5 canvas element. Finally, we demonstrated that the HTML5 canvas element, through our performance test, has a faster rendering process than SVG.

# Chapter 5

# Conclusions and Future Work

This chapter explains how the research issues of this work have been solved, and discusses the advantages and disadvantages in using the Glypher3D library, as well as points out in what aspects this library can be improved in the near future.

## 5.1 Conclusions

This dissertation shows that it is possible to create a Web3D graphics library without the obligation of installing any type of browser plug-ins or add-ons. In a way this means that the Glypher3D library provides an easier and more direct way of creating and interacting with 3D contents directly on browser. Also, by not depending on these plug-ins or add-ons, it differentiates itself from all the Web3D technologies mentioned on Chapter 2. On the other hand, this also means that Glypher3D its a cross platform API in terms of operating systems and browser technologies.

By combining graphics algorithms encoded in JavaScript with the HTML5 canvas element drawing and rendering functions, it is a library accessible to those with some knowledge of JavaScript and HTML. To use this library it is only necessary to set up the canvas tag and to call the library functions within a script in the HTML page (as shown in Chapter 3). The HTML5 canvas element, proved to be a good technology to draw the Glypher3D scenes within the canvas area, because it provides a good and simpler (than SVG) set of drawing functions, even at the level of individual pixels, and because has better rendering performance than SVG, as shown in Figure 4.8. However, because it does not provide scene DOM, interacting with the objects (for example, with mouse events) can be a complex task. Because the canvas tag it is just an HTML tag, any Glypher3D scene is full integrated on the web page.

The main drawback of the HTML5 canvas element, and also of the Glypher3D library, is its incompatibility with Internet Explorer, however, it is compatible with all the other major browsers, like Firefox, Safari, Opera and Chrome. To take advantage of all the HTML5 canvas element functions, it is recommended to use the latest browsers versions. This is due to the fact of the HTML5 markup language is a recent technology and is still in development. So, to take full advantage of the Glypher3D library, we recommend the following browsers

and versions: Firefox (3+), Safari (4+), Opera (10+) and Chrome (2+). To make the HTML5 canvas element accessible (only the simple drawing functions) to Internet Explorer, the ExplorerCanvas plug-in was developed by Google (see Chapter 3).

All the technologies mentioned in Chapter 2, take advantage of software or hardware acceleration (OpenGL or Directx), what allows them to provide fluid and high detailed 3D environments. On the other hand, Glypher3D does not use any plug-in to access the mentioned graphical engines, and due to the fact that all its algorithms are encoded in JavaScript (processed on the client machine), it has slower rendering performance than those Web3D technologies, especially when rendering complex objects with Gouraud shading (the only shading algorithm with pixel-based calculations).

However, when rendering simple objects, with color or shading algorithms that do not use the HTML5 canvas element pixel-based manipulation methods directly, Glypher3D is effective and the rendering is fast and fluid. The access to an web page with an Glypher3D scene is instantaneous and direct. On the other hand, a web page with other Web3D technologies (plug-ins or add-ons dependent) have higher loading times. With the results obtained in Chapter 4, it is possible to conclude that the Glypher3D library has a better performance on the Safari 4 browser, and that the rendering performance of an Glypher3D scene using flat shaded objects is much better than an Glypher3D scene using Gouraud shaded objects. Therefore, it can be said that Glypher3D is a good tool to directly draw and manipulate simple 3D scenes on browser. However, it cannot provide the level of realism and rendering speed of the studied Web3D technologies yet.

Only recently, we were advised about the existence of another Web3D library called OpenJSGL [33] with the same purposes and technologies (plug-in free through the HTML5 canvas element) of Glypher3D. However, while this library replicates the facilities of OpenGL, in order to ease the programming process to OpenGL developers, Glypher3D aims at creating graphics algorithms from scratch, but as simple and intuitive as possible, providing a different way in the creation and manipulation of an 3D scene. Notice that when this dissertation was proposed, we were unaware of the existence of OpenJSGL library. Besides, with the rapid growth of the Internet and browser capabilities, new technologies to deliver 3D contents over the web, without the obligation to install browser plug-ins or add-ons, are emerging at a rapid rate.

## 5.2 Future Work

Glypher3D can be improved in some aspects. For now, it does not allow to load or import models from external files, as it happens with other Web3D technologies. Consequently, all the models have to be created by programming in JavaScript. Importing models would save a lot of time, especially in the creation of very complex models. Also, the use of text, textures and a dynamic viewer position, would provide another level of realism to the Glypher3D scenes. We believe that the Gouraud shading algorithm can be improved to provide a better visualization and rendering speed of 3D scenes.

Glypher3D is a library that takes advantage of the HTML5 canvas element 2D rendering context to draw 3D scenes. However, it is expected that in the future a canvas 3D rendering

context will be introduced in HTML. With this 3D context accessible to JavaScript, developers may create 3D contents without the need of the additional libraries that nowadays add the third dimension to the web.

Also, very recently, it was announced by the Khronos Group (consortium that develops OpenGL standard) that efforts are being made to create a new 3D standard for the web without browser plug-ins. For that, it was formed the WebGL workgroup to define a JavaScript binding to OpenGL ES 2.0, which will make possible for browsers to render 3D contents rapidly without plug-ins.

# Bibliography


[1] Adobe. Actionscript Technology Center. http://www.adobe.com/devnet/actionscript/, 2009.

[2] Adobe. Exploring the New 3D Features in Flash CS4 Professional. http://www.adobe.com/devnet/flash/articles/3d_support.html, 2009.

[3] Adobe. Flash Lite: Features. http://www.adobe.com/products/flashlite/features/, 2009.

[4] Adobe. Flash Player 10: Complete Features. http://www.adobe.com/products/flashplayer/features/all_features/, 2009.

[5] Adobe. What is the Difference Between Shockwave and Flash. http://kb2.adobe.com/cps/139/tn_13971.html, 2009.

[6] Ajax3D. Ajax3d.org - Showcases and Tutorials. http://www.ajax3d.org/content/t2/indexa.html, 2009.

[7] Dave Astle and Kevin Hawkins. *Beginning OpenGL Game Programming*. Cengage Learning, 2004.

[8] Chris Bishop, Mark Paruzel, Andor Salga, Andrew Smith, and Cathy Leung. Canvas 3D JS Library. http://www.c3dl.org/, 2009.

[9] Don Brutzman and Leonard Daly. *X3D: Extensible 3D Graphics for Web Authors*. Morgan Kaufmann, 2007.

[10] Don Brutzman and Leonard Daly. X3D: Extensible 3D Graphics Standard. *IEEE Signal Processing Magazine*, pages 130–135, November 2007.

[11] Mozilla Development Center. The Canvas Element - Basic Usage. https://developer.mozilla.org/en/Canvas_tutorial/, 2009.

[12] Luca Chittaro and Roberto Ranon. Web3D Technologies in Learning, Education and Training: Motivations, Issues, Opportunities. In *Computers & Education*, pages 3–18. Elsevier Science Ltd, June 2007.

[13] Sophie 3D Engine. Sophie 3D Player - Flash 3D Engine. http://www.sophie3d.com/, 2009.

[14] Kenneth C. Finney. *3D Game Programming All in One*. Cengage Learning, 2004.

[15] Flash and Math. Simple 3D Dynamic Drawing in Flash CS4. http://www.flashandmath.com/flashcs4/cs4simple3d/index.html, 2009.

[16] James D. Foley, Andries Van Dam, Steven K. Feiner, and John F. Hughes. *Computer Graphics: Principles and Practice*. Addison-Wesley, 1995.

[17] Google. Explorercanvas HTML5 Canvas for Internet Explorer. http://code.google.com/p/explorercanvas/, 2009.

[18] Google. O3D Demos and Samples. http://code.google.com/intl/pt-PT/apis/o3d/docs/samplesdirectory.html, 2009.

[19] Google. O3D Technical Overview. http://code.google.com/intl/pt-PT/apis/o3d/docs/techoverview.html, 2009.

[20] Khronos Group. COLLADA - 3D Asset Exchange Schema. http://www.khronos.org/collada/, 2009.

[21] Khronos Group. OpenGL - The Industry's Foundation for High Performance Graphics. http://www.khronos.org/opengl/, 2009.

[22] Xichun Jennifer Guo. Phong Shading and Gouraud Shading. http://www.nbb.cornell.edu/neurobio/land/OldStudentProjects/cs490-95to96/guo/report.html, 1996.

[23] HTML5. The Canvas Element. http://www.whatwg.org/specs/web-apps/current-work/multipage/the-canvas-element.html, 2009.

[24] Java.Net. Java3D Enigma Cipher Machine Applet. https://enigma.dev.java.net/enigma-applet.html, 2009.

[25] Nigel W. John. The Impact of Web3D Technologies on Medical Education and Training. In *Computers & Education*, pages 19–31. Elsevier Science Ltd, June 2007.

[26] Neal Leavitt. Browsing the 3D Web. *IEEE Computer Society Press*, pages 22–24, September 2006.

[27] Catherine Leung, Andor Salga, and Andrew Smith. Canvas 3D JS Library. In *Proceedings of the 2008 Conference on Future Play: Research, Play, Share*, pages 274–275. ACM, November 2008.

[28] Francisco Luengo, Mariela Contreras, Aurely Leal, and Andres Iglesias. Interactive 3D Graphics Applications Embedded in Web Pages. In *Proceedings of the Computer Graphics, Imaging and Visualization*, pages 434–440. IEEE Computer Society, August 2007.

[29] Jeffrey J. McConnell. *Computer Graphics: Theory Into Practice*. Jones and Bartlett Publishers, 2004.

[30] Microsoft. DirectX: Advanced Graphics on Windows. `http://msdn.microsoft.com/en-us/directx/default.aspx`, 2009.

[31] David R. Nadeau. Building Virtual Worlds with VRML. *IEEE Computer Graphics and Applications*, pages 18–29, April 1999.

[32] Octaga. Octaga VRML and X3D Examples. `http://www.octaga.com/`, 2009.

[33] OpenJSGL. `http://openjsgl.sourceforge.net/openjsgl/doc/index.html`, 2009.

[34] Tony Parisi. Ajax3D: The Open Platform for Rich 3D Web Applications. *Whitepaper. Media Machines, Inc.*, August 2006.

[35] 3D Technologies RD. 3D Markup Language for Web - Open Source Platform for Creating 3D and 2D Interactive Web Content. `http://www.3dmlw.com/`, 2008.

[36] Chad F. Salisbury, Steven D. Farr, and Jason A. Moore. Web-based Simulation Visualization using Java3D. In *Proceedings of the 31st Conference on Winter simulation*, pages 1425–1429. ACM, January 1999.

[37] Sun. Java Plug-in Technology. `http//java.sun.com/products/plugin`, 2009.

[38] Vladimir Vukicevic. SVG and Canvas: Graphics for Web Apps. `http://people.mozilla.com/~vladimir/xtech2006/`, 2006.

[39] Vladimir Vukicevic. Canvas 3D: GL Power, Web-style. `http://blog.vlad1.com/2007/11/26/canvas-3d-gl-power-web-style/`, 2007.

[40] Vladimir Vukicevic. Canvas 3D. `http://blog.vlad1.com/canvas-3d/`, 2008.

[41] Vladimir Vukicevic. Canvas 3D Extension Update. `http://blog.vlad1.com/2009/03/28/canvas-3d-extension-update-2/`, 2009.

[42] W3C. Document Object Model (DOM). `http://www.w3.org/DOM/`, 2009.

[43] Aaron E. Walsh and Mikael Bourges-Sevenier. *Core WEB 3D*. Prentice Hall PTR, 2000.

# Appendix A

# Glossary

In this appendix is given the description of frequently used abbreviations.

**3DMLW** 3D Markup Language for Web

**3DS** 3ds Max

**Ajax** Asynchronous JavaScript and XML

**API** Application Programming Interface

**ASCII** American Standard Code for Information Interchange

**AVM** ActionScript Virtual Machine

**C3DL** Canvas 3D JS Library

**COLLADA** COLLAborative Design Activity

**CSS** Cascading Style Sheet

**DOM** Document Object Model

**DIS** Distributed Interactive Simulation

**H-Anim** Humanoid Animation

**HTML** HyperText Markup Language

**ISO** International Organization for Standardization

**JIT** Just In Time compiler

**JRE** Java Runtime Environment

**JVM** Java Virtual Machine

**MOCAP** Motion Capture

**NURBS** Non-Uniform Rational Bézier Spline

**RIA** Rich Internet Applications

**SAI** Scene Authoring Interface

**SVG** Scalable Vector Graphics

**VML** Vector Markup Language

**VRML** Virtual Reality Markup Language

**W3C** World Wide Web Consortium

**WWW** World Wide Web

**X3D** eXtensible 3D

**XML** eXtensible Markup Language

# Appendix B

# Web3D Technologies Resources

In this appendix are listed some online resources for the Web3D technologies mentioned in Chapter 2.

## B.1 VRML and X3D

### B.1.1 Players

The VRML-only most common web browsers plug-ins available are:

- Cosmo Player. `http://ovrt.nist.gov/cosmo`
- Cortona3D. `http://www.cortona3d.com/cortona`

Because X3D is an evolution of VRML, most of the VRML models (even models created 10 years ago) run in the newest X3D players. For X3D and VRML, the most common web browsers plug-ins and/or standalone players are:

- Octaga Player. `http://www.octaga.com`
- Vivaty Player (formerly Flux Player). `http://vivaty-player.software.informer.com`
- BS Contact. `http://www.bitmanagement.com/download/`
- SwirlX3D. `http://www.pinecoast.com/swvdownload.htm`
- Free WRL. `http://freewrl.sourceforge.net`
- Open VRML. `http://openvrml.org`

The X3D-only most common web browser plug-ins and/or standalone players are:

- Xj3D. `http://www.xj3d.org/`
- Instant Player. `http://www.instantreality.org/downloads/`

### B.1.2 Authoring Tools

The most common VRML and X3D authoring tools are:

- X3D-Edit. `http://savage.nps.edu/X3D-Edit`
- Octaga Modeler. `http://www.octaga.com`
- Vivaty Studio. `http://www.vivaty.com/downloads/studio`
- BS Editor. `http://www.bitmanagement.com/download/BS_Editor`
- Wings3D. `http://www.wings3d.com`
- Blender. `http://www.blender.org`

### B.1.3 Online Resources

For more VRML and X3D players, authoring tools, books, examples, etc, visit:

- `http://www.web3d.org/x3d/content/examples/`

To check if of your operating system and your web browser are compatibles with the chosen VRML or X3D players, and to check which players are open source or not, visit:

- `http://cic.nist.gov/vrml/vbdetect.html`

For VRML and X3D tutorials and examples visit the following websites:

- `http://x3dgraphics.com/examples/X3dForWebAuthors`
- `http://www.web3d.org/x3d/content/examples`
- `http://www.web3d.org/x3d/vrml/objects/oblib.htm`
- `http://web3dart.org`
- `http://sourceforge.net/projects/x3d`
- `http://hypermultimedia.com/x3d/Examples`
- `http://downloads.xj3d.org/webstart`
- `http://www.ocnus.com/models/PlatonicSolids`
- `http://vrmlworks.crispen.org/models.html#Models`
- `http://www.pacranch.com/vrmlresources/indexmodels.html`
- `http://www.rccad.com/Gallery_NonFlying.htm`
- `http://www.agocg.ac.uk/train/vrml2rep/part1/guide3.htm`
- `http://www.sdsc.edu/~moreland/courses`

## B.2 Java3D

### B.2.1 Online Resources

To download the Java3D API, visit:

- `http://java.sun.com/javase/technologies/desktop/java3d`

The Java Runtime Environment and Java Plug-in is available at:

- `http://java.sun.com/javase/downloads/index.jsp`

For Java3D online examples, visit:

- `https://java3d.dev.java.net`
- `http://www.apas.com/pzt/java3d/index.html`
- `http://www.java3d.org`

To help you in Java3D programming visit:

- `http://java.sun.com/developer/onlineTraining/java3d`
- `http://www.java3d.org/tutorial.doc`
- `http://www.javaworld.com/javaworld/jw-12-1998/jw-12-media.html?page=1`
- `http://www.tecgraf.puc-rio.br/~ismael/Cursos/Cidade_CG/labs/Java3D`
- `http://java3d.j3d.org/tutorials/index.html`

## B.3 Flash 3D

### B.3.1 Flash engines

Flash 2.5D AS3 isometric engines:

- Ffilmation. `http://www.ffilmation.org/website`
- Alternativa Platform. `http://blog.alternativaplatform.com/en`

Flash 3D engines:

- Papervision 3D. `http://www.papervision3d.org`
- Electric 3D. `http://www.electricoyster.com`

- Away 3D. `http://away3d.com`
- Sophie 3D. `http://www.sophie3d.com/website/index_en.php`
- Alternativa Platform. `http://blog.alternativaplatform.com/en`
- Sandy 3D. `http://www.flashsandy.org`
- Wire Engine 3D. `http://osflash.org/we3d`

Flash 3D game engines:

- PaperWorld 3D. `http://paperworld3d.com`
- SWFZ. `http://drawlogic.com/category/swfz`

Flash 3D vector engine:

- Five 3D. `http://five3d.mathieu-badimon.com`

Flash 3D physics engine:

- WOW Engine. `http://seraf.mediabox.fr/wow-engine`

3D Flash components:

- FreeSpin 3D. `http://www.freespin3d.com/Home.aspx`

Flash 3D animation framework:

- Cast 3D. `http://www.cast3d.com`

Flash 3D modeler:

- G-Nero. `http://www.g-nero.com`

### B.3.2 Online Resources

To download the Flash Player 10 visit:

- `http://get.adobe.com/br/flashplayer/`

For proprietary and open-source Flash authoring tools visit:

- `http://www.adobe.com/br/products/flash`
- `http://osflash.org/open_source_flash_projects`

Some Flash 3D examples, as well as full Flash 3D web environments can be seen at:

- http://www.ecodazoo.com
- http://www.onemotion.com/flash/spider
- http://strille.net/works/as3/raytracer
- http://blog.alternativaplatform.com/en/
- http://www.electricoyster.com/electric3d/index.html

Some Flash 3D tutorials are available at:

- http://theflashblog.com/?cat=13
- http://www.adobe.com/devnet/flash/3d_animation.html
- http://www.flashandmath.com/flashcs4
- http://www.tutorialized.com/tutorials/Flash/3D/1
- http://www.kirupa.com/developer/actionscript/3dindex.htm
- http://www.flashmagazine.com/Tutorials/detail/flash_3d_basics
- http://www.senocular.com/flash/tutorials/as3withflashcs3

## B.4 C3DL

### B.4.1 Online Resources

The download of the Mozilla Firefox Canvas 3D add-on is available at:

- https://addons.mozilla.org/en-US/firefox/addon/7171

For extra information about the experimental Canvas 3D add-on to Firefox, visit:

- http://groups.google.com/group/canvas-3d
- http://blog.vlad1.com/category/mozilla/canvas-3d
- http://people.mozilla.com/~vladimir/canvas3d/docs/
- http://people.mozilla.com/~vladimir/canvas3d/examples/
- https://labs.mozilla.com/forum/?CategoryID=9

The C3DL library, as well as tutorials, development news, documentation and examples are available at C3DL home page:

- http://www.c3dl.org

## B.5 Ajax3D

### B.5.1 Online Resources

Detailed information about Ajax3D, as well as examples, tutorials and discussion forums, are available at Ajax3D home page:

- `http://www.ajax3d.org`

## B.6 3DMLW

### B.6.1 Online Resources

To download the 3DMLW web browser plug-in, to see more detailed information, tutorials and examples of 3DMLW, visit:

- `http://www.3dmlw.com`

To edit 3DMLW documents, visit the 3DMLW editor page:

- `http://www.quantumhog.com`

## B.7 O3D

### B.7.1 Online Resources

The O3D web browser plug-in is available at:

- `http://tools.google.com/dlpage/o3d`

For samples and detailed information about O3D visit:

- `http://code.google.com/intl/pt-PT/apis/o3d`
- `http://google-code-updates.blogspot.com`
- `http://groups.google.com/group/o3d-discuss`
- `http://o3d.blogspot.com`
- `http://www.o3dexamples.com`

To create 3D models you can use:

- `http://sketchup.google.com`