

**TRƯỜNG ĐẠI HỌC KHOA HỌC TỰ NHIÊN  
KHOA CÔNG NGHỆ THÔNG TIN  
LỚP CỬ NHÂN TÀI NĂNG**

**NGUYỄN CÔNG THÀNH - BÙI TRẦN MẠNH HIẾU**

**NGHIÊN CỨU VÀ PHÁT TRIỂN  
FRAMEWORK GAME 3D TRÊN NỀN TẢNG  
HTML5**

**KHÓA LUẬN TỐT NGHIỆP CỬ NHÂN CNTT**

**TP.HCM, 2012**

**TRƯỜNG ĐẠI HỌC KHOA HỌC TỰ NHIÊN  
KHOA CÔNG NGHỆ THÔNG TIN  
LỚP CỬ NHÂN TÀI NĂNG**

**NGUYỄN CÔNG THÀNH      0812473  
BÙI TRẦN MẠNH HIẾU      0812139**

**NGHIÊN CỨU VÀ PHÁT TRIỂN  
FRAMEWORK GAME 3D TRÊN NỀN TẢNG  
HTML5**

**KHÓA LUẬN TỐT NGHIỆP CỬ NHÂN TIN HỌC**

**GIÁO VIÊN HƯỚNG DẪN  
ThS. LƯƠNG VĨ MINH – NGUYỄN ĐỨC HUY**

**NIÊN KHÓA 2008 – 2012**

## **NHẬN XÉT CỦA GIÁO VIÊN HƯỚNG DẪN**

## Khóa luận đáp ứng yêu cầu của LV cử nhân tin học.

TpHCM, ngày ..... tháng ..... năm 2012

Giáo viên hướng dẫn

## **NHẬN XÉT CỦA GIÁO VIÊN PHẢN BIỆN**

## Khóa luận đáp ứng yêu cầu của LV cử nhân tin học.

TpHCM, ngày ..... tháng ..... năm 2012

## Giáo viên phản biện

## LỜI CẢM ƠN

Chúng em xin chân thành cảm ơn Khoa Công Nghệ Thông Tin, trường Đại Học Khoa Học Tự Nhiên, Tp.HCM đã tạo điều kiện tốt cho chúng em thực hiện đề tài này.

Chúng em xin chân thành cảm ơn Thầy Trần Minh Triết, Thầy Lương Vĩ Minh và Thầy Nguyễn Đức Huy là những người đã tận tình hướng dẫn, chỉ bảo chúng em trong suốt thời gian thực hiện đề tài.

Chúng em cũng xin gửi lời cảm ơn sâu sắc đến quý Thầy Cô trong Khoa đã tận tình giảng dạy, trang bị cho chúng em những kiến thức quý báu trong những năm học vừa qua.

Chúng em xin gửi lòng biết ơn sâu sắc đến Ba, Mẹ, các anh chị và bạn bè đã ủng hộ, giúp đỡ và động viên chúng em trong những lúc khó khăn cũng như trong suốt thời gian học tập và nghiên cứu.

Mặc dù chúng em đã cố gắng hoàn thành luận văn trong phạm vi và khả năng cho phép, nhưng chắc chắn sẽ không tránh khỏi những thiếu sót, kính mong sự cảm thông và tận tình chỉ bảo của quý Thầy Cô và các bạn.

Nhóm thực hiện

# ĐỀ CƯƠNG CHI TIẾT

<b>Tên Đề Tài:</b> Nghiên cứu và phát triển framework game 3D trên nền tảng HTML5
<b>Giáo viên hướng dẫn:</b> ThS. Lương Vĩ Minh – Nguyễn Đức Huy
<b>Thời gian thực hiện:</b> từ ngày 06/02/2012 đến ngày 09/07/2012
<b>Sinh viên thực hiện:</b>
Nguyễn Công Thành (0812473) – Bùi Trần Mạnh Hiếu (0812139)
<b>Loại đề tài:</b> Tìm hiểu công nghệ và xây dựng ứng dụng
<b>Nội Dung Đề Tài</b> (mô tả chi tiết nội dung đề tài, yêu cầu, phương pháp thực hiện, kết quả đạt được, ...):  Đây là đề tài thuộc về hướng tìm hiểu công nghệ và phát triển ứng dụng. Đề tài bao gồm các phần sau: <ul style="list-style-type: none"><li>Khảo sát các công nghệ làm Game 3D trên nền tảng Web bao gồm HTML5 – WebGL, Silverlight 5 , Adobe Flash.</li><li>Nghiên cứu công nghệ HTML5 - WebGL</li><li>Tìm hiểu các vấn đề liên quan đến đồ họa 3D với WebGL.</li><li>Xác định, phân tích các vấn đề và đề ra các giải pháp để phát triển framework game 3D cho dòng game nhập vai với công nghệ WebGL.</li><li>Xây dựng thử nghiệm ứng dụng game 3D trên nền tảng Web và thiết bị di động sử dụng game framework được xây dựng.</li></ul>

**Kế Hoạch Thực Hiện:**

- 06/02/2012-11/02/2012: Khảo sát các công nghệ làm Game 3D trên Web
- 12/02/2012-22/02/2012: Mô tả các đặc trưng chính của dòng game sẽ xây dựng.
- 23/02/2012-09/03/2012: Nêu và phân tích các vấn đề sẽ phát sinh trong quá trình xây dựng.
- 10/03/2012-10/04/2012: Đưa ra các giải pháp cho các vấn đề và lựa chọn giải pháp thích hợp.
- 11/04/2012-15/05/2012: Xây dựng game framework.
- 16/05/2012-15/06/2012: Xây dựng game minh họa trên nền tảng Web sử dụng game framework đã xây dựng.
- 16/06/2012-01/07/2012: Xây dựng game trên thiết bị di động sử dụng game framework đã xây dựng.
- 02/07/2012-09/07/2012: Cài tiến các ứng dụng game.

<b>Xác nhận của GVHD</b>	<b>Ngày tháng năm 2012</b> <b>Nhóm SV Thực hiện</b>
	<b>Nguyễn Công Thành – Bùi Trần Mạnh Hiếu</b>

# MỤC LỤC

Chương 1 Mở đầu.....	1
1.1. Game 3D .....	1
1.2. Web 3D .....	2
1.3. Khảo sát hiện trạng .....	4
1.3.1. Nhu cầu của việc xây dựng game 3D trên nền tảng Web.....	4
1.3.2. Khảo sát công nghệ WebGL.....	4
1.3.3. Khảo sát công nghệ Silverlight 5.....	6
1.3.4. Khảo sát công nghệ Adobe Flash .....	7
1.3.5. Đánh giá các công nghệ 3D trên Web .....	8
1.3.6. Kết luận.....	9
1.4. Mục tiêu đề tài.....	9
1.5. Nội dung luận văn .....	10
Chương 2 Tổng quan về WebGL .....	12
2.1. Giới thiệu về WebGL.....	12
2.2. Các vấn đề liên quan việc tạo lập context và drawing buffer trong WebGL .....	15
2.2.1. Canvas.....	15
2.2.2. Drawing Buffer.....	17
2.2.3. WebGL Viewport .....	17
2.3. Các loại biến sử dụng trong WebGL .....	18
2.4. Các phương thức của đối tượng WebGL context .....	20
2.4.1. Đối tượng Shader .....	20

2.4.2. Đổi tượng Program .....	20
2.4.3. Biến Attribute .....	20
2.4.4. Biến Uniform .....	20
2.4.5. Đổi tượng Buffer.....	20
2.4.6. Textures .....	20
2.4.7. Misc .....	21
<b>Chương 3 Các vấn đề cơ bản liên quan đến đồ họa 3D trong WebGL.....</b>	<b>22</b>
3.1. Hệ trục toạ độ 3D dùng trong WebGL.....	22
3.2. Lập trình điều khiển GPU .....	23
3.2.1. Luồng xử lý của GPU .....	23
3.2.2. Cài đặt các hàm xử lý GLSL .....	26
3.2.3. Sử dụng mã lệnh GLSL trong mã nguồn WebGL .....	28
3.2.4. Sử dụng Texture trong WebGL .....	30
3.3. Tương tác của người dùng trong ứng dụng game 3D WebGL .....	32
3.4. Kiến trúc cơ bản của game 3D WebGL .....	33
3.5. Kiến trúc quản lý resources game .....	35
3.6. Kiến trúc liên quan đến lập trình điều khiển GPU - Effect.....	37
3.7. Vẽ đa giác trong lập trình 3D với WebGL.....	38
<b>Chương 4 Các vấn đề liên quan đến mô hình 3D .....</b>	<b>42</b>
4.1. Giới thiệu về đặc trưng mô hình 3D .....	42
4.2. Load mô hình 3D .....	44
4.3. Vẽ mô hình 3D .....	46
4.4. Sử dụng mô hình 3D trong code .....	49

4.5.	Thay đổi tỷ lệ mô hình .....	49
4.6.	Animation của Mô hình 3D .....	51
4.7.	Kiến trúc sử dụng mô hình 3D trong ứng dụng. ....	53
	Chương 5 Các vấn đề nâng cao về đồ họa 3D trong WebGL .....	55
5.1.	Hiệu ứng camera .....	55
5.1.1.	Cơ bản về camera .....	55
5.1.2.	Chế độ camera theo sau đối tượng.....	58
5.2.	Tạo dựng địa hình .....	59
5.2.1.	Tạo địa hình từ một ảnh.....	59
5.2.2.	Lợp texture trên địa hình .....	63
5.2.3.	Chiếu sáng địa hình .....	65
5.2.4.	Kỹ thuật Skybox .....	67
5.3.	Kỹ thuật Blending .....	68
5.4.	Kỹ thuật Billboard.....	71
5.5.	Kỹ thuật Particle.....	74
5.6.	Kỹ thuật Ray Picking .....	76
5.7.	Xử lý va chạm .....	78
5.8.	Hỗ trợ đa người dùng .....	81
	Chương 6 Kiến trúc xử lý kịch bản và màn chơi trong game .....	85
6.1.	Kiến trúc xử lý kịch bản.....	85
6.1.1.	Cấu trúc file kịch bản.....	86
6.1.2.	Kiến trúc module xử lý kịch bản .....	87
6.2.	Kiến trúc quản lý màn chơi.....	89

6.2.1.	Quản lý Building.....	89
6.2.2.	Quản lý Character .....	90
6.2.3.	Quản lý Map .....	92
6.3.	Kiến trúc quản lý thuộc tính trong game .....	94
6.4.	Xử lý việc hiển thị lời thoại .....	96
<b>Chương 7</b>	<b>Kết luận .....</b>	<b>98</b>
7.1.	Các kết quả đạt được .....	98
7.2.	Một số hình ảnh minh họa trong game .....	100
7.2.1.	Game Devil :.....	100
7.2.2.	Game Devil phiên bản Mobile :.....	107
7.2.3.	Game GunPlay :.....	109
7.2.4.	Game GunPlay phiên bản Mobile : .....	114
7.3.	Hướng phát triển của đề tài.....	115

## **DANH MỤC CÁC HÌNH**

Hình 1-1 Minh họa lịch sử phát triển game .....	2
Hình 1-2 Trang Web dạy học sinh học theo dạng 3D.....	3
Hình 1-3 Mạng xã hội 3D IMVU.....	3
Hình 1-4 Iphone4S - Ipad2 chạy WebGL sử dụng Safari .....	5
Hình 1-5 Kiến trúc tổng quan công nghệ WebGL .....	5
Hình 1-6 Kiến trúc tổng quan của Silverlight 5 .....	7
Hình 1-7 Kiến trúc tổng quan Adobe Flex .....	8
Hình 2-1 Mô hình kiến trúc WebGL .....	13
Hình 2-2 Giai đoạn một của quá trình vẽ .....	14
Hình 2-3 Giai đoạn hai của quá trình vẽ .....	15
Hình 2-4 Các bước xử lý trong WebGL.....	19
Hình 3-1 Toạ độ theo quy tắc bàn tay trái và bàn tay phải .....	22
Hình 3-2 Sơ đồ luồng xử lý của GPU .....	24
Hình 3-3 Các bước vẽ hình 3D trong WebGL .....	26
Hình 3-4 Sơ đồ luồng xử lý của ứng dụng game .....	35
Hình 3-5 Lớp Effect .....	37
Hình 4-1 Mô hình nhân vật 3 chiều.....	43
Hình 4-2 Mô hình JSON .....	45
Hình 4-3 Kết quả vẽ một mô hình 3D.....	49
Hình 4-4 Mô hình không có animation .....	51
Hình 4-5 Mô hình có animation .....	51
Hình 4-6 Kiến trúc sử dụng mô hình 3D .....	53

Hình 5-1 Minh họa up vector trong Camera .....	56
Hình 5-2 Minh họa fieldOfView và aspectRatio .....	57
Hình 5-3 Minh họa nearPlaneDistance và farPlaneDistance .....	57
Hình 5-4 Camera theo sau đối tượng .....	59
Hình 5-5 Ảnh height map.....	60
Hình 5-6 Lưới địa hình.....	60
Hình 5-7 Các khối của địa hình.....	61
Hình 5-8 Thứ tự vẽ các đỉnh trong một khối.....	62
Hình 5-9 Tọa độ texture lợp địa hình .....	63
Hình 5-10 Khoảng độ cao của địa hình .....	64
Hình 5-11 Địa hình được lợp bởi nhiều texture .....	65
Hình 5-12 Nguồn sáng song song .....	66
Hình 5-13 Vector pháp tuyến tổng .....	66
Hình 5-14 Chiếu sáng địa hình.....	67
Hình 5-15 Các texture trong skybox .....	68
Hình 5-16 Minh họa vấn đề trong kỹ thuật Blending .....	69
Hình 5-17 Kỹ thuật Blending tạo hình ảnh trong suốt .....	71
Hình 5-18 Kỹ thuật billboard .....	72
Hình 5-19 Kỹ thuật Billboard.....	74
Hình 5-20 Kỹ thuật Particle tạo hiệu ứng lửa quanh nhân vật.....	75
Hình 5-21 Minh họa RayPicking thể hiện ở 2 kiểu Camera .....	76
Hình 5-22 Minh họa gốc tọa độ click chuột trước và sau khi chuyển đổi. ....	77
Hình 5-23 Minh họa BoundingBox của một mô hình 3D.....	79

Hình 5-24 Cấu trúc của BoundingBox.....	79
Hình 5-25 Minh họa 2 Object 3D trước và sau khi va chạm .....	80
Hình 5-26 Minh họa 2 Object 3D trước và sau khi va chạm theo chiều X, Z .....	81
Hình 5-27 Chức năng Multiplayer trong Game (Devil).....	82
Hình 5-28 Minh họa Chat giữa các người chơi với nhau trong Game (GunPlay) ....	84
Hình 6-1 Sơ đồ các lớp PostAction thực thi hành động của Event.....	87
Hình 6-2 Sơ đồ lớp module quản lý kịch bản .....	88
Hình 6-3 Quản lý building .....	90
Hình 6-4 Quản lý Character .....	92
Hình 6-5 Quản lý Map .....	93
Hình 6-6 Quản lý thuộc tính.....	95
Hình 6-7 Xử lý lời thoại trong game .....	97
Hình 7-1 Chơi game GunPlay bằng điện thoại LG-GT540 Android 2.3 .....	100
Hình 7-2 Màn hình chọn nhân vật.....	101
Hình 7-3 Màn hình bắt đầu chơi game .....	101
Hình 7-4 Người chỉ đường hướng dẫn nhân vật sang thế giới Devil.....	102
Hình 7-5 Trò chuyện với NPC .....	102
Hình 7-6 Sang thế giới Devil bằng Teleport .....	103
Hình 7-7 Cảnh đánh quái vật ở thế giới Devil .....	103
Hình 7-8 Bảng thông tin nhân vật .....	104
Hình 7-9 Chưởng phép bằng cách click chuột lên màn hình .....	105
Hình 7-10 Chiêu thứ 3 : hào quang xuất hiện quanh nhân vật.....	105
Hình 7-11 Nhìn ở góc nhìn khác .....	106

Hình 7-12 Chuẩn bị đấu với Devil khổng lồ .....	106
Hình 7-13 Hiện thông báo khi đánh thắng Devil khổng lồ .....	107
Hình 7-14 Chơi ở chế độ Multiplayer .....	107
Hình 7-16 Màn hình Menu chọn phe .....	109
Hình 7-17 Bắt đầu vào cảnh chơi.....	109
Hình 7-18 Quan sát khung cảnh với góc nhìn đứng trên đồi núi .....	110
Hình 7-19 Hai phe bắn nhau .....	110
Hình 7-20 Đối phương phát hiện và bắn nhân vật của mình .....	111
Hình 7-21 Một quân bị chết .....	111
Hình 7-22 Màn hình Menu khi bị bắn hết máu .....	112
Hình 7-23 Thay đổi màn chơi sang Map khác .....	112
Hình 7-24 Hai người chơi thấy nhau trong một màn chơi .....	113
Hình 7-25 Chat với nhau trong Game ở chế độ Multiplayer .....	113
Hình 7-26 Menu chính của Game .....	114
Hình 7-27 Chơi game ở chế độ SinglePlayer.....	114

## **DANH MỤC CÁC BẢNG**

Bảng 1-1 So sánh, đánh giá các công nghệ WebGL, Flash, Silverlight .....	9
Bảng 2-1 Bảng các giá trị mặc định của Drawing Buffer trong WebGL.....	17

## TÓM TẮT KHÓA LUẬN

Như chúng ta đã biết, nhu cầu giải trí của con người là rất quan trọng và được xem là một phần không thể thiếu trong cuộc sống. Để đáp ứng nhu cầu đó, game đóng vai trò không nhỏ. Từ xưa, con người đã có các hoạt động sinh hoạt thể thao trong những lúc rảnh rỗi nhằm rèn luyện thể chất, tinh thần giúp cho họ sống khỏe hơn, thông minh hơn và sáng tạo hơn. Game cũng là một trong các thể loại đó, game giúp con người rèn luyện sự kiên nhẫn, sự tiên đoán chính xác, tinh thần đồng đội và tính sáng tạo.

Cùng với sự mở rộng của Internet, sự phát triển công nghệ Web đã đem lại cho con người nhiều tiện ích. Sự kết hợp giữa game và công nghệ Web sẽ góp phần đem game đến với mọi người. Ngoài ra với khả năng sản xuất các game 3D trên Web sẽ mang đến những trải nghiệm hấp dẫn cho người chơi.

Nội dung đề tài chúng em tập trung vào việc nghiên cứu các công nghệ làm game 3D trên Web và xây dựng một framework game 3D để hỗ trợ phát triển game. Chúng em đã tìm hiểu, đưa ra cách giải quyết các vấn đề phát sinh trong khi thực hiện một game. Chúng em còn tìm hiểu cách phát triển game sao cho dễ dàng nâng cấp và thay đổi giúp cho game ngày càng hấp dẫn và phong phú hơn.

## **Chương 1**

### **Mở đầu**

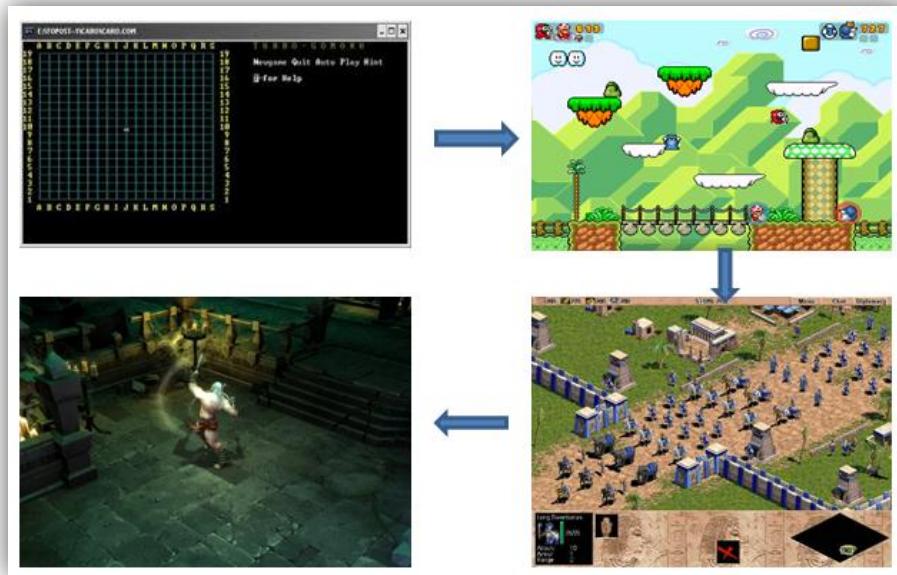
*Nội dung Chương 1 trình bày tổng quan về Game, nhu cầu xây dựng Game3D trên web và khảo sát các công nghệ làm game 3D trên Web hiện nay. Từ đó đề ra mục tiêu hướng tới của đề tài.*

#### **1.1. Game 3D**

Game được xem là một môn giải trí đã tồn tại từ khá lâu, từ game đơn giản nhất : đánh cờ ca rô trên giao diện console đơn sơ, mộc mạc. Để đáp ứng nhu cầu chơi game có hình ảnh đẹp hơn, các nhà phát triển đã xây dựng game dưới dạng 2D theo kiểu load image. Diễn hình là game Mario kinh điển,... Đối với thời điểm đó, để có máy tính mà chơi game Mario là sự ao ước của bao nhiêu thiếu nhi. Nhưng để thu hút người chơi ở độ tuổi thanh thiếu niên, cần phải có một bước vọt khác, xây dựng game sao cho có vẻ thật hơn, kịch bản hấp dẫn, đòi hỏi người chơi phải sử dụng đầu óc để chiến thắng. Từ đó, các thể loại game chiến thuật, nhập vai dạng 2.5D ra đời. Về mặt lý thuyết, nó được xem là 2D, nhưng người chơi nhìn có cảm giác thật hơn vì có chiều sâu, nhưng chưa phải 3D vì không thể quay góc nhìn 360°. Một số game 2.5D đình đám một thời : Empires, Consack, Tam Quốc Chí, Redalert, ...

Cuộc sống ngày càng phát triển, công nghệ ngày càng hiện đại, nhu cầu con người ngày càng tăng cao, các game thủ muốn chơi game có thể thật hơn, muốn nhập vai vào nhân vật để thực hiện nhiệm vụ trong game. Từ đó, các nhà phát triển không ngừng cố gắng và nỗ lực, phát triển game 2.5D của mình sang dạng 3D như game Dialo, Football, ... Người chơi có thể hòa vào thế giới 3D của nhân vật, có cảm giác thật khi chơi. Số lượng người chơi ngày càng tăng, vì thế, đòi hỏi phía lập trình có những ý kiến sáng tạo, luôn đổi mới game, để có thể thu hút người chơi.

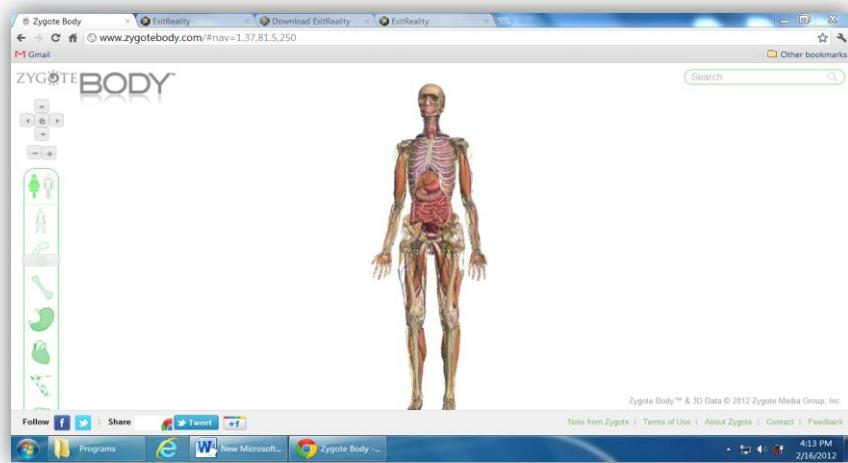
Công nghệ ngày càng phát triển, cộng với sự phát triển lớn mạnh của Internet, vì thế game được hỗ trợ kết nối Internet để các game thủ có thể chơi online với nhau thay vì chơi mạng LAN bình thường. Điều này cho ta thấy, nhu cầu phát triển game ngày càng lớn mạnh, đặc biệt là game 3D online.



**Hình 1-1 Minh họa lịch sử phát triển game**

## 1.2. Web 3D

Mới nghe qua từ Web3D, người nghe có thể nghĩ Web3D là một trang Web nhưng ở dạng 3D. Nhưng đó là cách nghĩ từ phía người sử dụng, đối với nhà phát triển, lập trình viên, có thể nghĩ Web3D là những mô hình 3D, vật thể 3D được đưa lên trên Web. Vậy tại sao lại có nhu cầu đưa những mô hình 3D này lên Web. Có rất nhiều câu trả lời cho câu hỏi này. Có thể phía phát triển muốn người sử dụng có cảm giác thật hơn, sự chuyển động 3D sẽ khiến người dùng cảm thấy hứng thú, thay vì những hình ảnh 2D tĩnh. Hoặc có thể do nhu cầu của các nhà đầu tư về dạy học anh văn, muốn trang web dạy anh văn thật hơn, có hình ảnh 3D chuyển động hấp dẫn để cuốn hút người đến trang Web. Thật vậy, đã có rất nhiều trang Web thể hiện nội dung 3D mới mục đích dạy học như hình 1.1, xây dựng mạng xã hội 3D như hình 1.2, ...



### Hình 1-2 Trang Web dạy học sinh học theo dạng 3D

Như vậy, nhu cầu phát triển ứng dụng, game 3D trên nền tảng web khá được quan tâm. Hiện tại có khá nhiều công nghệ hỗ trợ việc thể hiện 3D trên nền tảng Web. Lập trình viên sẽ sử dụng những công nghệ đó để phát triển, đưa trang web từ 2D sang dạng 3D. Nhưng những công nghệ đó, đối với thời điểm quá khứ, hiện tại, tương lai có điểm mạnh, điểm yếu khác nhau. Vì thế, để có thể lựa chọn công nghệ tốt nhất cho thời điểm hiện tại và hướng đến tương lai thì chúng ta phải khảo sát những công nghệ đó một cách trực quan.



### Hình 1-3 Mạng xã hội 3D IMVU

### **1.3. Khảo sát hiện trạng**

#### **1.3.1. Nhu cầu của việc xây dựng game 3D trên nền tảng Web**

Nhu cầu chơi game 3D đã được các nhà thiết kế game đáp ứng không chỉ về mặt đồ họa mà còn ở kịch bản game hấp dẫn, luôn đổi mới, khiến người chơi ngày càng hứng thú khi chơi. Nhưng những game đó cần phải cài đặt và yêu cầu máy có cấu hình mạnh và dung lượng lớn để chứa game. Như vậy, vấn đề đặt ra là có cách nào chơi game 3D không cần cài đặt, vừa đẹp, mà vừa mượt không. Từ đó, việc xây dựng game 3D trên nền tảng web được nảy sinh và hiện tại đã có nhiều trang web game 3D, mạng xã hội 3D, dạy học 3D và tương lai game 3D lên nền tảng Web sẽ phát triển rất mạnh.

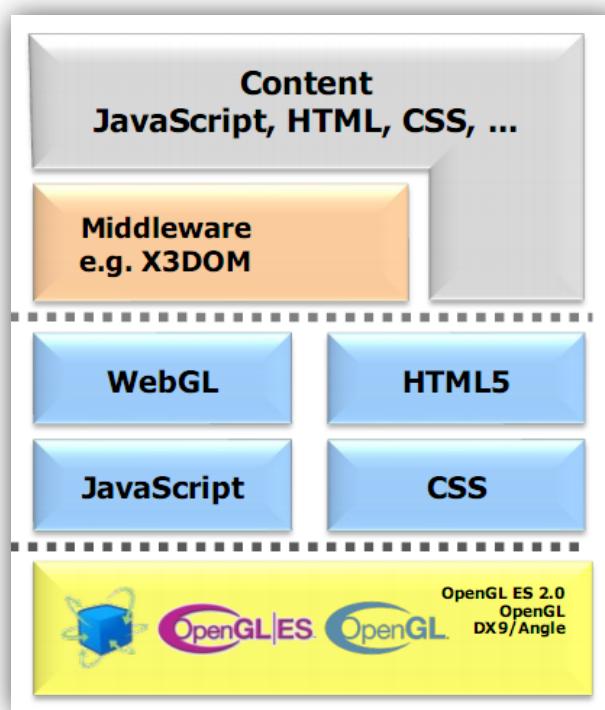
#### **1.3.2. Khảo sát công nghệ WebGL**

Tại hội thảo các nhà phát triển game năm 2011 (Game Developers Conference - San Francisco 3/3/2011), Khronos Group đã cho ra đời WebGL phiên bản 1.0 với sự khai thác tối đa đồ họa 3D của phần cứng trên trình duyệt hỗ trợ HTML5, mà không cần bất kỳ một plug-in nào. WebGL được viết trên ngôn ngữ Javascript, dựa trên OpenGL ES 2.0 cho phép sử dụng các API đồ họa 3D trong trình duyệt ở bất kì nền tảng nào hỗ trợ chuẩn OpenGL hay OpenGL ES. Các phiên bản mới nhất của các trình duyệt đều hỗ trợ WebGL (trừ Internet Explorer).

Về mảng thiết bị di động, hiện tại Nokia N900, Sony Ericsson Xperia 2011 đã hỗ trợ WebGL, chạy rất mượt nhờ phần cứng khá tốt, ngoài ra Opera đã ra mắt Opera Mobile phiên bản 12 hỗ trợ WebGL đối với Android 1.6<sup>+</sup>. Điện thoại Android 1.6<sup>+</sup> sau khi cài đặt Opera Mobile 12 có thể mở được các trang web sử dụng WebGL, nhưng tùy vào phần cứng, tốc độ, hình ảnh của các thiết bị sẽ khác nhau. Đối với thiết bị hệ điều hành iOS Iphone/Ipad, hiện tại trình duyệt Safari của Ipad3 đã hỗ trợ WebGL. Nhưng đối với Iphone4/Iphone4S/Ipad2 iOS 5.x phải dùng phần mềm để kích hoạt WebGL.



Hình 1-4 Iphone4S - Ipad2 chạy WebGL sử dụng Safari



Hình 1-5 Kiến trúc tổng quan công nghệ WebGL

❖ **Ưu điểm :**

- Là chuẩn web mới cho API đồ họa 3D cấp thấp trên nền OpenGL ES

- Các phiên bản mới của các trình duyệt đều hỗ trợ (trừ Internet Explorer)
- Không cần cài thêm bất kỳ Plug-in nào
- Lập trình trên ngôn ngữ Javascript, có thể lập trình hướng đối tượng
- Cross-Platform kể cả iOS
- Có thể sử dụng các thẻ của HTML5.
- Đã có nhiều Middleware ra đời hỗ trợ lập trình cho lập trình viên như : ThreeJS, SceneJS, SpiderGL, ... nhưng vẫn còn đang trong quá trình xây dựng và thử nghiệm.

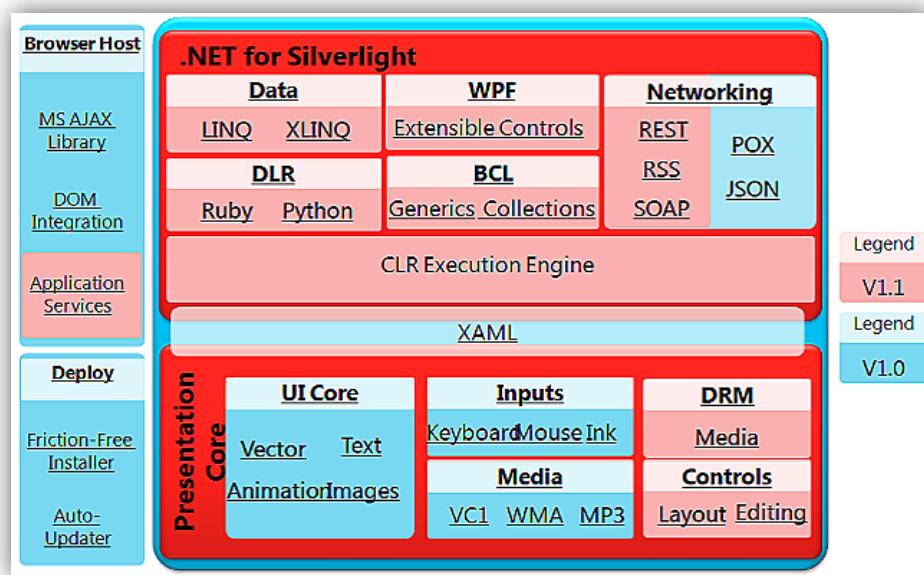
❖ **Khuyết điểm :**

- Chưa có IDE chuẩn để hỗ trợ cho các lập trình viên
- Có sự khác biệt lớn trong trình diễn 3D giữa các thiết bị có phần cứng khác nhau

### **1.3.3. Khảo sát công nghệ Silverlight 5**

Silverlight được phát hành phiên bản đầu tiên vào năm 2007 bởi tập đoàn Microsoft, đến nay đã phát hành đến phiên bản 5. Silverlight là một ứng dụng framework để viết và chạy các ứng dụng internet với sự đa phương tiện về hình ảnh và đồ họa. Nó là một dạng plug-in trên công nghệ của Microsoft .Net nó hỗ trợ Ajax, Python, Ruby và các ngôn ngữ lập trình. Net như Visual basic, C#. Silverlight thể hiện mức độ truyền tải âm thanh và hình ảnh chất lượng cao một cách nhanh chóng và hiệu quả trên các trình duyệt chính như Internet Explorer, Firefox, Safari.

Về mặt chạy trên nền tảng thiết bị di động bị hạn chế, chỉ chạy được trên Window Phone 7 và Symbian.



**Hình 1-6 Kiến trúc tổng quan của Silverlight 5**

❖ **Ưu điểm :**

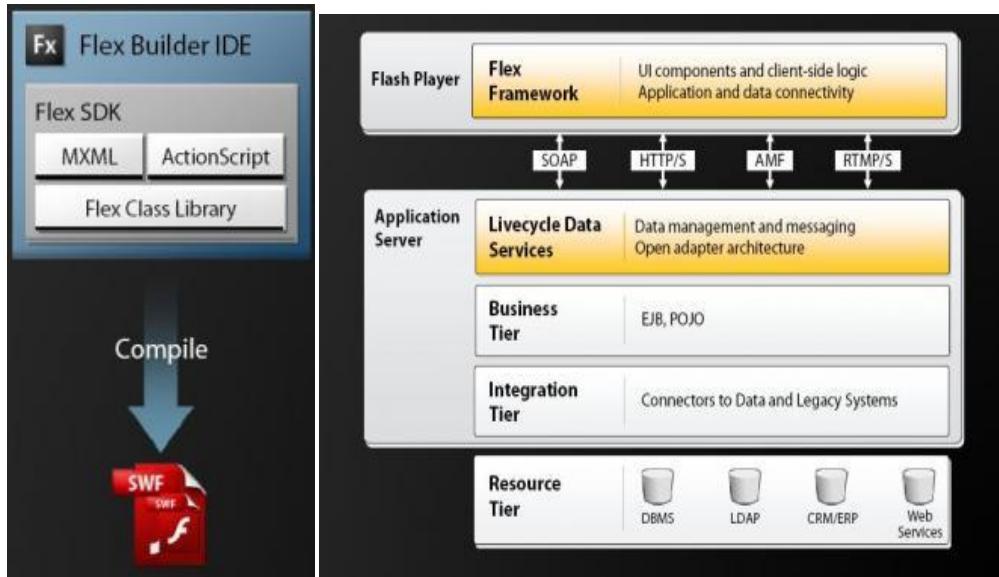
- Là một công nghệ hỗ trợ mạnh mẽ RIA.
- Có nhiều IDE mạnh : Visual Studio, Expression Blend.
- Lập trình trên các ngôn ngữ quen thuộc : C#, VB. net
- Tích hợp ngôn ngữ XAML hỗ trợ lập trình thiết kế giao diện

❖ **Khuyết điểm :**

- Cần phải cài Plug-in
- Chỉ chạy trên thiết bị di động là Window Phone 7, Symbian.
- Công cụ phát triển có tính phí.

#### **1.3.4. Khảo sát công nghệ Adobe Flash**

Adobe Flash/Flex là công nghệ mới của hãng Adobe cung cấp công nghệ phát triển Rich Internet Application (RIA) chuyên nghiệp. Flash được biết đến khá lâu trên thế giới và phiên bản hiện tại là Flex 3.0, tuy nhiên công nghệ này còn khá mới mẻ tại Việt Nam. Adobe Flex bao gồm Actionscript 3.0, MXML (Macromedia XML)CSS. Các công nghệ/công cụ liên quan bao gồm Flex SDK, Flex Builder, Flex Chartting.



**Hình 1-7 Kiến trúc tổng quan Adobe Flex**

❖ **Ưu điểm :**

- Rất quen thuộc với người dùng Internet, bởi plugin của Flash được cài đặt trong đa số các trình duyệt.
- Là một công nghệ hỗ trợ mạnh mẽ RIA.
- Ưu điểm lớn nhất của Flash - với đồ họa dạng vector - là kích thước file rất nhỏ. Thuận tiện cho việc truyền tải dữ liệu qua Internet.
- Công nghệ flash hỗ trợ rất tốt cho việc lập trình.

❖ **Khuyết điểm :**

- Chưa hỗ trợ trên iOS.
- Các tính năng về RIA hầu hết đã được hỗ trợ bởi HTML5.
- Cần phải cài plugin.
- Nếu chạy nhiều file Flash cùng một lúc trên một trang web thì rất nặng.

### 1.3.5. Đánh giá các công nghệ 3D trên Web

Sau đây là bảng so sánh các khía cạnh khác nhau của ba công nghệ phát triển ứng dụng web mạnh nhất hiện nay :

	<b>WebGL</b>	<b>Flash</b>	<b>SilverLight</b>
<b>Trình duyệt hỗ trợ</b>	Các phiên bản mới của Firefox,	Tất cả các trình duyệt	IE6+, Firefox 3+, Chrome, Safari

	Chrome, Opera, Safari		
<b>Hệ điều hành hỗ trợ</b>	Cross-platform với iOS, Android	Cross-platform (trừ iOS)	Window. Mac
<b>Ưu điểm</b>	Kế thừa từ OpenGL	Có công cụ mạnh hỗ trợ bao gồm sự tích hợp Unity3D	API thân thiện với lập trình viên
<b>Khuyết điểm</b>	Chưa có IDE	Khả năng tương thích với iOS, mất nhiều ưu thế trong HTML5	Là sản phẩm của Microsoft, chưa phải là chuẩn
<b>Plugin</b>	Không	Có	Có
<b>Kết luận</b>	Rất tốt. Thích hợp phát triển quảng cáo, xây dựng ứng dụng, game đa nền tảng	Tốt cho việc quảng cáo, xây dựng ứng dụng. Nhưng không được hỗ trợ trên thiết bị di động	Rất tốt, nhưng trong tương lai gần

**Bảng 1-1 So sánh, đánh giá các công nghệ WebGL, Flash, Silverlight**

### **1.3.6. Kết luận**

Từ việc khảo sát trên ta thấy, để tạo ra một ứng dụng game 3D trên nền tảng web thật sự có nhiều công nghệ hỗ trợ việc này. Mỗi công nghệ đều có những ưu, khuyết điểm khác nhau. WebGL được đánh giá cao, nó là một chuẩn mới của công nghệ Web3D. Lợi thế lớn nhất của nó là kế thừa từ OpenGL, một công nghệ rất mạnh về việc phát triển ứng dụng, game 3D và được các tên tuổi lớn chong lồng. Ngoài ra, các phiên bản mới của các trình duyệt hiện nay đều hỗ trợ WebGL (trừ Internet Explorer) và không cần cài đặt thêm một plugin nào.

## **1.4. Mục tiêu đề tài**

Đề tài này thuộc về hướng nghiên cứu và tìm hiểu công nghệ, từ đó xây dựng và phát triển ứng dụng. Mục tiêu của đề tài là nghiên cứu công nghệ **WebGL** và xây dựng thử nghiệm game 3D dựa trên công nghệ **WebGL**. Nội dung của đề tài bao gồm:

- Nghiên cứu kỹ thuật phát triển game 3D với công nghệ **WebGL**

- Xây dựng thử nghiệm một **framework game** thể loại game nhập vai từ các kiến thức đã tìm hiểu.
- Xây dựng các ứng dụng game, cảnh chơi để minh họa cho **game framework** đã viết.

Như vậy mục tiêu của đề tài là xây dựng một **framework game** trên nền tảng công nghệ **WebGL**.

## 1.5. Nội dung luận văn

Luận văn của chúng em bao gồm các nội dung sau :

### Chương 01 : Mở đầu

Nội dung chương 01 trình bày tổng quan về Game, nhu cầu xây dựng Game3D trên web, khảo sát các công nghệ 3D trên nền tảng web. Từ đó đề ra mục tiêu hướng tới của đề tài.

### Chương 02 : Tổng quan về WebGL

Giới thiệu tổng quan kiến trúc của WebGL, các vấn đề liên quan tạo lập context và drawing buffer trong WebGL, các bước cơ bản trong quá trình hiển thị các đối tượng 3D trong WebGL.

### Chương 03 : Các vấn đề cơ bản liên quan đến đồ họa 3D trong WebGL

Nội dung chương này đề cập đến các vấn đề cơ bản về đồ họa 3D trong WebGL.

### Chương 04 : Các vấn đề liên quan đến mô hình 3D

Trình bày các vấn đề về tạo lập, quản lý và xử lý những hành động của nhân vật trong không gian ba chiều trên môi trường WebGL.

### Chương 05 : Các vấn đề nâng cao về đồ họa 3D trong WebGL

Trình bày các vấn đề nâng cao về đồ họa 3D trong WebGL

## **Chương 06 : Kiến trúc xử lý kịch bản và màn chơi trong game**

Trình bày các vấn đề về kiến trúc xử lý kịch bản, quản lý màn chơi, quản lý các thuộc tính và thể hiện lời thoại trong game.

## **Chương 07 : Kết luận**

Trình bày các kết quả đạt được trong luận văn và hướng phát triển của đề tài

## Chương 2

# Tổng quan về WebGL

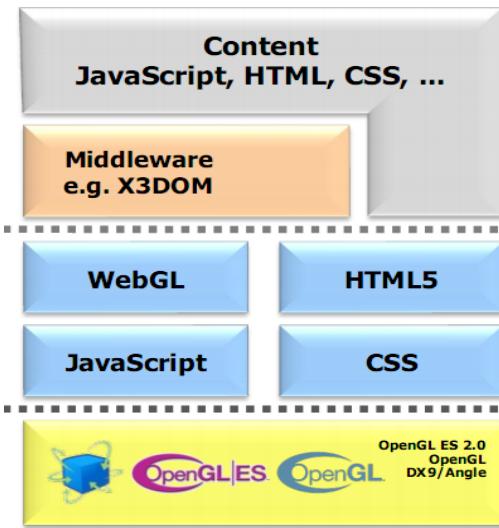
*Nội dung chương này sẽ giới thiệu tổng quan kiến trúc của WebGL, các vấn đề liên quan tạo lập context và drawing buffer trong WebGL, các bước cơ bản trong quá trình hiển thị các đối tượng 3D.*

### 2.1. Giới thiệu về WebGL

WebGL là một API xử lý đồ họa 3D được dùng trên môi trường web. WebGL được phát triển dựa trên nền tảng OpenGL ES và cung cấp đầy đủ những chức năng tương tự OpenGL ES. Điểm mạnh của WebGL là khả năng phát triển các ứng dụng 3D cross-platform. Một ứng dụng 3D WebGL có thể bao gồm các file HTML, CSS, Javascript.

WebGL khai thác sức mạnh của phần cứng để tăng tốc xử lý đồ họa 3D. WebGL sử dụng OpenGL Shading Language (GLSL), một ngôn ngữ lập trình trên GPU, cho phép người lập trình kiểm soát được quá trình dựng hình 3D qua đó tăng hiệu suất thực thi và nâng cao chất lượng đồ họa.

Kết quả của quá trình vẽ hình 3D bằng WebGL được thể hiện thông qua đối tượng canvas HTML5. Hiện tại WebGL có thể chạy trực tiếp mà không cần cài đặt plugin trên hầu hết các trình duyệt web HTML5 như Google chrome, mozilla firefox, Safari, Opera.



**Hình 2-1 Mô hình kiến trúc WebGL**

(Nguồn: [http://www.khronos.org/assets/uploads/developers/library/2011\\_GDC\\_WebGL/WebGL-Intro\\_GDC-Mar11.pdf](http://www.khronos.org/assets/uploads/developers/library/2011_GDC_WebGL/WebGL-Intro_GDC-Mar11.pdf))

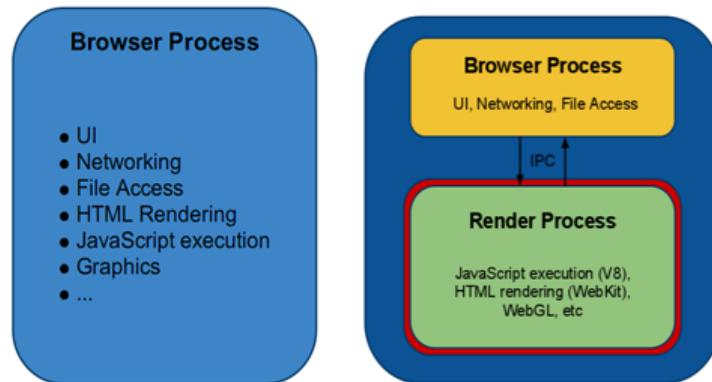
Kiến trúc của WebGL được chia ra làm ba tầng cơ bản như sau:

- ❖ Tầng thứ nhất bao gồm các nội dung đã được tải xuống từ internet. Middleware WebGL là những thành phần cho phép những người không phải là chuyên gia về lập trình 3D vẫn có khả năng lập trình với WebGL. Một middleware tiêu biểu là X3DOM. Đây là một framework cho phép hiển thị nội dung 3D dưới dạng các thẻ HTML.
- ❖ Tầng thứ hai là tầng chức năng WebGL và các thành phần khác như CSS, HTML5, Javascript.
- ❖ Tầng thứ ba là các Driver được cung cấp bởi hệ điều hành.

Dưới đây là các mô hình mô tả một cách chi tiết luồng xử lý của WebGL để hiện thị lên các đối tượng 3D. Có hai giai đoạn chính của quá trình hiển thị một khung nhìn 3D.

- ❖ Giai đoạn thứ nhất đó là sự liên lạc giữa quá trình duyệt và quá trình vẽ. Ở quá trình duyệt thì trình duyệt sẽ nhận các dữ liệu đầu vào như giao diện người dùng(UI), các gói tin mạng, file, các mã Javascript... Tiếp đó các dữ liệu này sẽ được truyền qua cho quá trình vẽ bằng kênh truyền Inter-Process

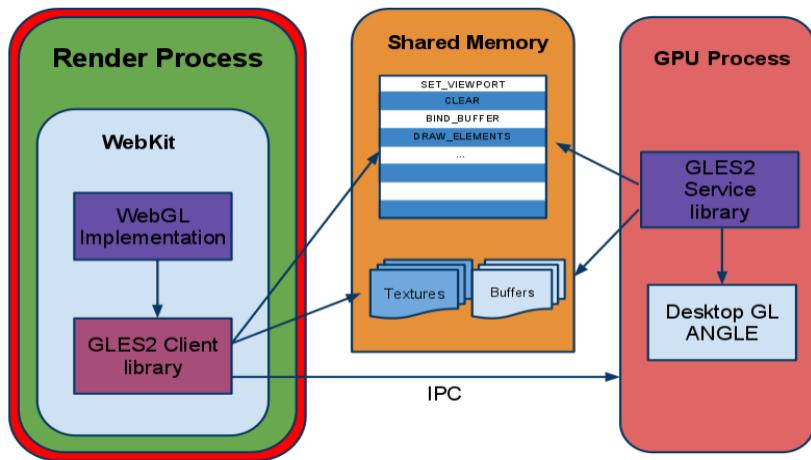
Communication(IPC). Đây là kênh truyền cung cấp một tập hợp các phương thức cho phép truyền nhận các dữ liệu giữa các Thread hay các Process. Các phương thức được chia làm các thành phần chính như truyền nhận các message, bắt đồng bộ, chia sẻ bộ nhớ và điều khiển các thủ tục.



**Hình 2-2 Giai đoạn một của quá trình vẽ.**

(Nguồn: [http://www.khronos.org/assets/uploads/developers/library/2010\\_siggraph\\_bof\\_webgl/WebGL-BOF-2-WebGL-in-Chrome\\_SIGGRAPH-Jul29.pdf](http://www.khronos.org/assets/uploads/developers/library/2010_siggraph_bof_webgl/WebGL-BOF-2-WebGL-in-Chrome_SIGGRAPH-Jul29.pdf))

- ❖ Hình dưới đây mô tả quá trình xử lý ở giai đoạn thứ hai, ở giai đoạn này, WebGL sẽ gọi các hàm GLES 2.0 tương ứng để xử lý dữ liệu được truyền từ trên xuống, sau đó sẽ tiến hành ghi các hàm GLES lên bộ nhớ dùng chung, rồi từ đó GPU của máy tính client dựa vào các hàm đã ghi trong bộ nhớ dùng chung đó để tính toán để vẽ các hình ảnh 3D lên các buffer, nếu máy tính client không hỗ trợ thư viện GLES 2.0 thì GLES service sẽ dùng thư viện Google Angle để chuyển các gọi các hàm Direct X tương ứng để vẽ. Sau cùng các trình duyệt sẽ lấy các buffer này và vẽ lên đối tượng canvas đã được định nghĩa sẵn.



**Hình 2-3 Giai đoạn hai của quá trình vẽ**

(Nguồn: [http://www.khronos.org/assets/uploads/developers/library/2010\\_siggraph\\_bof\\_webgl/WebGL-BOF-2-WebGL-in-Chrome\\_SIGGRAPH-Jul29.pdf](http://www.khronos.org/assets/uploads/developers/library/2010_siggraph_bof_webgl/WebGL-BOF-2-WebGL-in-Chrome_SIGGRAPH-Jul29.pdf))

## 2.2. Các vấn đề liên quan việc tạo lập context và drawing buffer trong WebGL

### 2.2.1. Canvas

Canvas là một thành phần mới được giới thiệu trong HTML5. Canvas được sử dụng thông qua thẻ `<canvas>`. Canvas được dùng để hiển thị đồ họa bằng các đoạn lệnh javascript. Thông thường chúng ta thường quan tâm đến 2 thuộc tính cơ bản của canvas là width và height. Dưới đây là một ví dụ sử dụng canvas trong chương trình.

```

<html>
  <body>
    <canvas id="canvas" width="800" height="600"></canvas>
  </body>
</html>

```

Để sử dụng được WebGL API trước tiên phải lấy được một đối tượng `WebGLRenderingContext` từ một canvas [1]. Đối tượng `WebGLRenderingContext` được tạo bằng lời gọi là `getContext()` với tham số là chuỗi ‘experimental-webgl’. Khi được gọi lần đầu tiên, một đối tượng `WebGLRenderingContext` sẽ được tạo ra và trả về, đồng thời vùng đệm dùng để vẽ (drawing buffer) được tạo ra. Những lời

gọi hàm getContext() với cùng chuỗi tham số như trên sẽ trả về cùng một đối tượng. Dưới đây là đoạn mã nguồn minh họa việc lấy context từ đối tượng canvas.

```
<script type="text/javascript">
    var canvas;
    var gl;
    function Init() {
        canvas = document.getElementById("canvas");
        gl = canvas.getContext("experimental-webgl");
    }
</script>
```

Tham số thứ hai có thể được truyền vào getContext() đó là một đối tượng WebGLContextAttributes chứa những tham số cấu hình được dùng để khởi tạo drawing buffer [1]. Tuy nhiên tham số này là không bắt buộc và nếu những cấu hình này không được hỗ trợ bởi phần cứng đồ họa thì nó sẽ không gây ra lỗi khi lấy context. Chúng ta có thể truy vấn WebGLContextAttributes thông qua lời gọi hàm getContextAttributes() của đối tượng WebGLRenderingContext.

Bên cạnh khả năng vẽ 3D, thành phần canvas còn có thể được sử dụng để hiển thị hình ảnh 2D. Khi đó đối tượng context được tạo từ gọi hàm getContext() với chuỗi đầu vào là '2d'. Context 2D cung cấp các hàm cần thiết để vẽ đường thẳng, đường tròn, tô màu, vẽ ảnh, viết chữ ...

Dưới đây là một hàm vẽ một hình vuông trên canvas. Nên lưu ý tham số truyền vào hàm getContext trong trường hợp này là '2d'. Chúng ta không thể vẽ đồng thời hình 2D và 3D trong cùng một canvas.

```
function draw() {
    var canvas = document.getElementById('canvas');
    var context = canvas.getContext('2d');
    context.beginPath();
    context.moveTo(0,0);
    context.lineTo(10,0);
    context.lineTo(10,10);
    context.lineTo(0,10);
    context.closePath();
    context.stroke();
}
```

### **2.2.2. Drawing Buffer**

Drawing buffer là một vùng nhớ đệm, nơi những lời gọi hàm API thực hiện các thao tác vẽ hình [1]. Drawing buffer được tạo ra khi đối tượng WebGLRenderingContext được tạo. Bảng dưới đây liệt kê tất cả những buffer thuộc về drawing buffer cùng với kích thước nhỏ nhất. Kích thước của drawing buffer được xác định dựa vào thuộc tính width và height của thành phần canvas. Bảng dưới đây cũng cho thấy các giá trị mặc định khi mà những buffer này được làm sạch khi lần đầu tiên được tạo và khi kích thước nó thay đổi.

Buffer	Giá trị làm sạch	Kích thước nhỏ nhất	Được định nghĩa mặc định
Color	(0, 0, 0, 0)	8 bits cho mỗi thành phần	Có
Depth	1.0	16 bit integer	Có
Stencil	0	8 bits	Không

**Bảng 2-1 Bảng các giá trị mặc định của Drawing Buffer trong WebGL**

(Nguồn: [1] )

Nếu không thể cung cấp một drawing buffer có kích thước đúng theo width và height được yêu cầu thì một drawing buffer với kích thước nhỏ hơn sẽ được tạo ra [1]. Chúng ta có thể sử dụng đối tượng WebGLContextAttribute để thay đổi drawing buffer sau khi nó đã được định nghĩa. Ví dụ như WebGLContextAttribute có thể được sử dụng để xác định color buffer có chứa kênh alpha hay không, nếu có sử dụng kênh alpha, thì color buffer sẽ được kết hợp với phần còn lại của trang HTML.

### **2.2.3. WebGL Viewport**

Viewport được sử dụng trong WebGL là một hình chữ nhật. Khi WebGL context được tạo ra, viewport sẽ được khởi tạo là một hình chữ nhật với gốc là (0,0) width và height tương ứng là canvas.width và canvas.height [1]. Dưới đây là một ví dụ thiết lập viewport.

```
var canvas;  
var gl;
```

```

function Init() {
    canvas = document.getElementById("canvas");
    gl = canvas.getContext("experimental-webgl");
    gl.viewport(0, 0, canvas.width, canvas.height);
}

```

Để cập nhật tự động viewport, chúng ta cần phải xử lý sự kiện *resize* của canvas để cập nhật lại kích thước viewport.

### 2.3. Các loại biến sử dụng trong WebGL

Quá trình xử lý của WebGL bao gồm nhiều bước như vertex shader, primitive assembly, rasterization, fragment shader (xem Hình 2-4). Trong đó chúng ta đặc biệt quan tâm vertex shader và fragment shader.

Vertex shader có nhiệm vụ xác định tọa độ sau cùng của vertex sau khi thực hiện tính toán với các ma trận [2]. Đầu vào của vertex shader là các biến kiểu attribute và uniform. Biến attribute là những thông tin riêng của từng vertex ví dụ như tọa độ 3D, vị trí để lấy màu trên texture, vector pháp tuyến. Biến uniform là những thông tin dùng chung cho các vertex ví dụ như ma trận projection, ma trận view, ma trận world , tọa độ và độ lớn của nguồn sáng. Vertex shader output ra các biến kiểu varying.

Fragment shader là bước xác định màu tại các pixel [2]. Fragment shader nhận vào các biến kiểu uniform ví dụ như texture dùng để lấy màu và các biến kiểu varying output từ vertex shader.

Dưới đây là một đoạn code GLSL minh họa cách sử dụng các loại biến trong vertex shader và fragment shader.

```

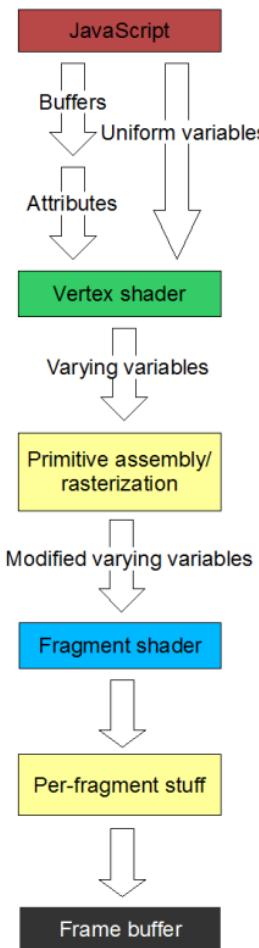
//Begin vertex shader
attribute vec3 aVertexPosition;
attribute vec2 aTextureCoord;
uniform mat4 uVMatrix;
uniform mat4 uPMatrix;
uniform mat4 uWMatrix;
varying vec2 vTextureCoord;
void main(void) {
    gl_Position      =      uPMatrix*uVMatrix*uWMatrix*vec4(aVertexPosition,
1.0);
}

```

```

    vTextureCoord = aTextureCoord;
}
//Begin fragment shader
varying vec2 vTextureCoord;
uniform sampler2D uSampler;
void main(void) {
    gl_FragColor      = texture2D(uSampler,      vec2(vTextureCoord.s,
vTextureCoord.t));
}

```



**Hình 2-4 Các bước xử lý trong WebGL**

(*Nguồn: [2]* )

## **2.4. Các phương thức của đối tượng WebGL context**

### **2.4.1. Đối tượng Shader**

- createShader : Tạo một đối tượng shader, tham số truyền vào là VERTEX\_SHADER hoặc FRAGMENT\_SHADER.
- shaderSource : Truyền code GLSL vào đối tượng shader
- compileShader : Biên dịch một đối tượng shader.
- attachShader : Attach một đối tượng shader vào một đối tượng program

### **2.4.2. Đối tượng Program**

- createProgram : Tạo một đối tượng program
- linkProgram : Link một đối tượng program
- useProgram : Khai báo sử dụng đối tượng program

### **2.4.3. Biến Attribute**

- getAttribLocation : Lấy địa chỉ của một biến attribute
- vertexAttribPointer : Định nghĩa một mảng attribute data cho các vertex.

### **2.4.4. Biến Uniform**

- getUniformLocation : Lấy địa chỉ của một biến uniform
- uniformMatrix4fv : Truyền giá trị cho một biến ma trận kiểm uniform

### **2.4.5. Đối tượng Buffer**

- createBuffer : Tạo một đối tượng buffer
- bindBuffer : Khai báo sử dụng đối tượng buffer.
- bufferData : Truyền dữ liệu cho một đối tượng buffer. (Cần gọi hàm bindBuffer trước để xác định buffer nhận dữ liệu)

### **2.4.6. Textures**

- createTexture : Tạo texture
- bindTexture : Khai báo sử dụng texture. Tương tự như bindBuffer , chúng ta cần gọi hàm bindTexture trước khi thao tác với texture.

#### **2.4.7. Misc**

- viewport : Xác định viewport
- clear : Xóa drawing buffers. Tham số là COLOR\_BUFFER\_BIT, DEPTH\_BUFFER\_BIT, STENCIL\_BUFFER\_BIT tương ứng với color, depth, stencil buffer
- clearColor : Xác định màu xóa màn hình
- drawArrays : Thực hiện thao tác vẽ từ một mảng vertex. Tham số là POINTS , LINES, LINE\_LOOP, LINE\_STRIP, TRIANGLES, TRIANGLE\_STRIP, TRIANGLE\_FAN
- drawElements : Thực hiện thao tác vẽ một mảng vertex theo chỉ số
- enable | disable : Tham số là BLEND, DEPTH\_TEST ...

### **2.5. Kết luận**

WebGL là một chuẩn mới của công nghệ 3D trên Web, nó được hỗ trợ bởi hầu hết bởi các phiên bản của các trình duyệt hiện nay, ngoại trừ Internet Explorer. WebGL có được một lợi thế rất lớn đó là kế thừa từ OpenGL, một công nghệ rất mạnh cho việc phát triển các ứng dụng 3D trên máy tính để và rất quen thuộc với các lập trình viên, đặc biệt là nó được sự hỗ trợ từ các tên tuổi lớn. Một điểm hết sức quan trọng đó là vì nó được cung cấp sẵn nên người dùng cuối không cần phải cài bất kỳ plugin nào cả để có thể chạy được các ứng dụng 3D. Năm được những kiến thức cơ bản về WebGL là một bước quan trọng để ta có thể thực hiện các hiệu ứng đồ họa cao cấp.

## Chương 3

# Các vấn đề cơ bản liên quan đến đồ họa 3D trong WebGL

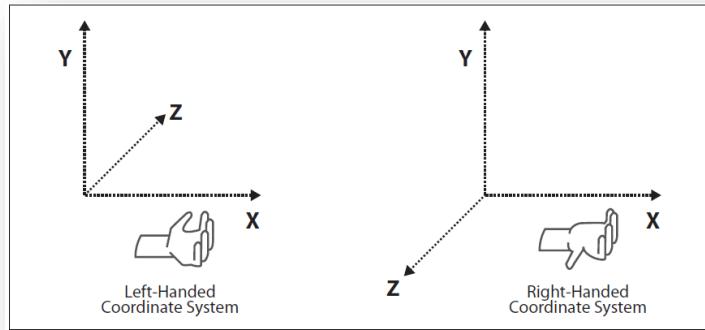
☞ Nội dung chương này đề cập đến các vấn đề cơ bản về đồ họa 3D trong WebGL

## 3.1. Hệ trục tọa độ 3D dùng trong WebGL

### 💡 Vấn đề

Úng với hướng của trục x và y cố định, ta có 2 khả năng cho hướng của trục z vuông góc với mặt phẳng tạo bởi trục x và y. Từ đó ta có 2 kiểu khác nhau của hệ tọa độ 3D theo quy ước “bàn tay phải” và “bàn tay trái”.

Phát biểu quy tắc: Đặt bàn tay sao cho chiều từ cổ tay ra các ngón tay là chiều trục x, co các ngón tay lên vuông góc ta có chiều các ngón tay là chiều trục y, khi đó chiều ngón cái duỗi thẳng ra là chiều trục z. Bàn tay nào thoả quy ước đó thì hệ tọa độ đó theo quy tắc bàn tay đó.



Hình 3-1 Toạ độ theo quy tắc bàn tay trái và bàn tay phải

(Nguồn: [3] )

### 💡 Giải pháp

WebGL quy ước hệ trục tọa độ 3D theo quy tắc bàn tay phải.

## 3.2. Lập trình điều khiển GPU

### Vấn đề

Để đạt được một số hiệu ứng đồ họa 3D, chúng ta phải lập trình điều khiển trực tiếp GPU cụ thể là vertex shader và fragment shader (còn được gọi là pixel shader). Việc lập trình shader cho phép ta kiểm soát được toàn bộ quá trình dựng hình đồng thời tận dụng được khả năng tính toán cực mạnh của GPU và giảm tải công việc rất nhiều cho CPU.

GLSL (OpenGL Shading Language) là ngôn ngữ lập trình cho GPU cũng là giải pháp của vấn đề này.

### Giải pháp

Để sử dụng được WebGL bắt buộc chúng ta phải có khả năng lập trình với GLSL. GLSL giúp tăng hiệu suất game và nâng cao chất lượng đồ họa. Mọi vertex được vẽ sẽ được truyền qua vertex shader, và mọi pixel được vẽ sẽ được truyền qua fragment shader. Những shaders này có thể thực hiện bất cứ thao tác nào trên dữ liệu đã truyền cho chúng.

### 3.2.1. Luồng xử lý của GPU

#### (1) Buffers:

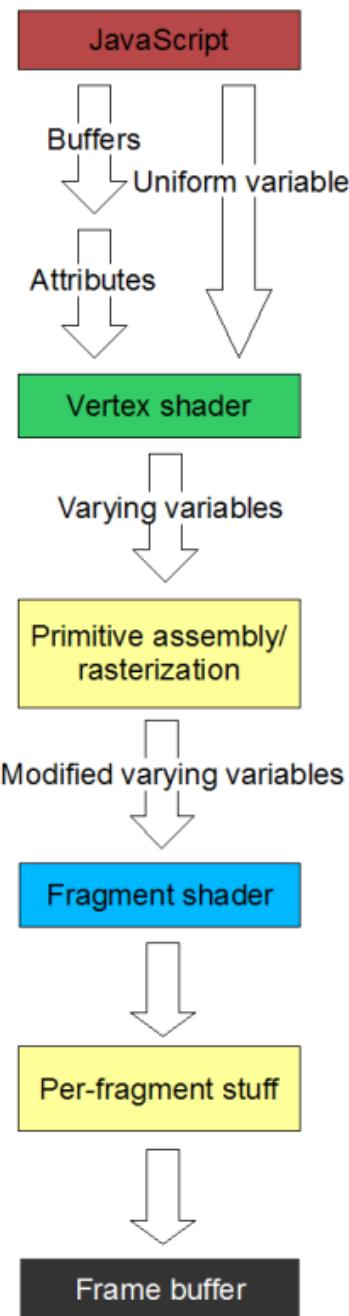
Theo sơ đồ trong Hình 3-2, ứng dụng WebGL sẽ truyền xuống cho Vertex Shader các dòng buffers. Buffer chính là một vùng nhớ lưu trữ một mảng các số thực. Khi vẽ bất kỳ một đối tượng 3D chúng ta sẽ cần đến một số loại buffer như buffer vị trí (position), buffer tọa độ texture, buffer vector pháp tuyến ... Ví dụ để vẽ 3 vertex thì buffer vị trí sẽ lưu trữ một mảng gồm 9 số tương ứng tọa độ x, y, z của từng vertex.

#### (2) Attributes:

Attributes là những thông tin riêng biệt của từng vertex được truyền vào vertex shader. Ví dụ như một mảng vector 3 thành phần mô tả vị trí của các vertex, một mảng vector 2 thành phần chứa thông tin tọa độ texture của các vertex ...

#### (3) Uniforms variable:

Khác hẳn với attributes, uniforms variable là những biến dùng chung cho các vertex trong quá trình vẽ. Ví dụ như ma trận projection, ma trận view, texture (các vertex có tọa độ lấy màu texture khác nhau trên cùng một texture) ...



**Hình 3-2 Sơ đồ luồng xử lý của GPU**

(*Nguồn: [2]* )

#### (4) Vertex Shader

Nhiệm vụ chính của Vertex shader là nhận vào một vertex với tọa độ trong không gian 3D, sau đó thực hiện phép nhân tọa độ này với các ma trận để xác định được vị trí kết quả và gán cho biến gl\_Position sau đó gởi giá trị này cùng những thông tin khác của vertex như tọa độ texture xuống Fragment Shader để xử lý tiếp.

#### (5) Varying Variables

Varying variable là biến được output từ vertex shader và là input cho fragment shader ví dụ như tọa độ lấy màu texture.

#### (6) Fragment Shader

Fragment shader là một bước được gọi cho mỗi pixel cần được vẽ. Trong hầu hết mọi trường hợp, nhiệm vụ của fragment shader là nhận dữ liệu input từ vertex shader và các biến uniform nếu có sau đó tính toán và đưa ra output là màu sắc của pixel.

#### (7) Primitive Assembly / Rasterization

Chức năng của Primitive Assembly là nối các vertex riêng lẻ thành những hình cơ sở (primitive) như đoạn thẳng, tam giác v.v...

Rasterization là quá trình xác định tất cả các pixel nằm trên một đoạn thẳng hoặc là nằm trong một tam giác. Rasterization đảm bảo xác định đầy đủ tất cả các pixels cần được tô màu.

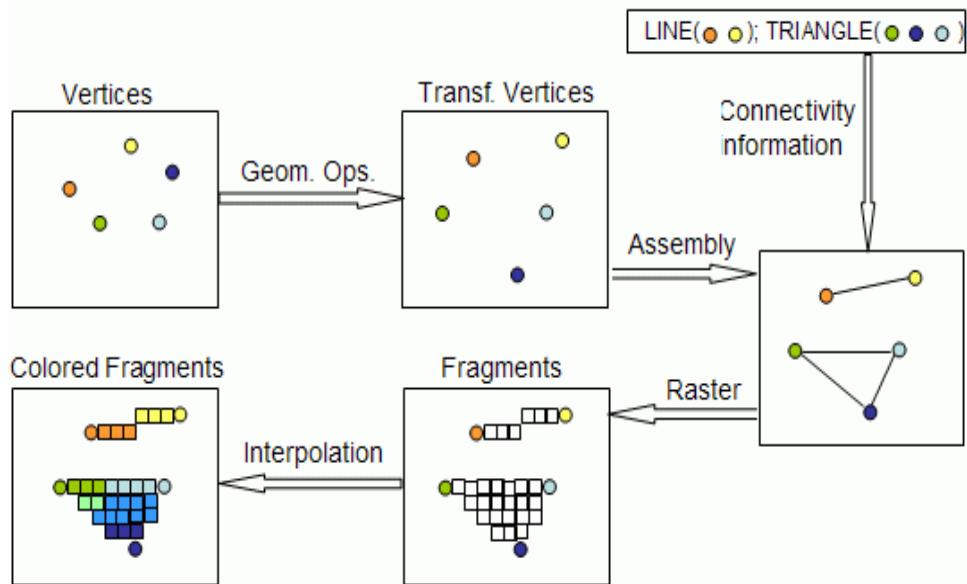
#### (8) Per-fragment

Các pixel được thêm vào sau quá trình rasterization cần được xác định màu. Phần per-fragment sẽ tính toán giá trị màu của pixel từ giá trị màu của các điểm góc tam giác bằng phép nội suy.

#### (9) Frame Buffer

Frame Buffer là nơi các phép render được blend với nhau trong trường hợp alpha blending.

Hình dưới đây minh họa các bước vẽ hình 3D, từ những vertex ban đầu cho đến khi tất cả các pixel đã được xác định màu.



**Hình 3-3 Các bước vẽ hình 3D trong WebGL**

(Nguồn <http://www.lighthouse3d.com/tutorials/glsl-tutorial/pipeline-overview/>)

### 3.2.2. Cài đặt các hàm xử lý GLSL

#### 1. Vertex Shader

Đầu tiên, cần phải định nghĩa các biến input và biến output của vertex shader

```
attribute vec3 aVertexPosition;
attribute vec2 aTextureCoord;
uniform mat4 uVMatrix;           //View Matrix
uniform mat4 uPMatrix;           //Projection Matrix
uniform mat4 uWMatrix;           //World Matrix
varying vec2 vTextureCoord;      //Output
```

Kế đến ta định nghĩa hàm xử lý Vertex Shader

```
void main(void) {
    gl_Position = uPMatrix*uVMatrix*uWMatrix*vec4(aVertexPosition, 1.0);
    vTextureCoord = aTextureCoord;
}
```

gl\_Position là biến bắt buộc của Vertex Shader. Trong ví dụ trên ta tính giá trị cho gl\_Position và gán giá trị cho biến output vTextureCoord.

#### 2. Fragment Shader

Trong fragment shader ta cần xác định độ chính xác của phép tính số thực [2]. Ở ví dụ dưới đây ta xác định độ chính xác mức trung bình.

```
precision mediump float;
```

Cũng như Vertex Shader, trước khi đi vào cài đặt Fragment Shader ta cần định nghĩa các biến input cho Fragment Shader.

```
varying vec2 vTextureCoord;  
uniform sampler2D uSampler;
```

Fragment Shader nhận vào biến output từ Vertex Shader và các biến uniform với những thông tin này Fragment Shader sẽ thực hiện tính toán để xác định giá trị màu tại pixel.

```
void main(void) {  
    gl_FragColor=texture2D(uSampler,vec2(vTextureCoord.s,vTextureCoord.t));  
}
```

gl\_FragColor là biến bắt buộc trong Fragment shader. Trong ví dụ trên ta dùng hàm texture2D để xác định màu tại vị trí có tọa độ vTextureCoord (biến output của vertex shader) trên texture uSampler

### 3.2.3. Sử dụng mã lệnh GLSL trong mã nguồn WebGL

#### 1. Khởi tạo Shaders

Mã nguồn GLSL là dạng text thông thường. Ta khởi tạo các đối tượng shaders (xem 2.4). Lưu ý gl là một biến toàn cục kiểu WebGLRenderingContext (xem 2.2.1)

```
var vertexShader = gl.createShader(gl.VERTEX_SHADER);
gl.shaderSource(vertexShader, vertexShaderCode);
gl.compileShader(vertexShader);

var fragmentShader = gl.createShader(gl.FRAGMENT_SHADER);
gl.shaderSource(fragmentShader, fragmentShaderCode);
gl.compileShader(fragmentShader);
```

Trong ứng dụng, ta sẽ khởi tạo một đối tượng program để quản lý shaders

```
var shaderProgram = gl.createProgram();
gl.attachShader(shaderProgram, vertexShader);
gl.attachShader(shaderProgram, fragmentShader);
gl.linkProgram(shaderProgram);
```

Có thể có nhiều đối tượng program trong cùng một ứng dụng, để sử dụng một program cụ thể ta dùng hàm useProgram

```
gl.useProgram(shaderProgram);
```

Chúng ta cần phải xác định địa chỉ của các biến kiểu attributes và uniforms bằng cách sử dụng hàm getAttribLocation và getUniformLocation [2]. Trong ví dụ dưới đây chúng ta lưu địa chỉ thành các property của shaderProgram như shaderProgram.vertexPositionAttribute tuy nhiên bản thân object program không có property vertexPositionAttribute mà chúng ta đang sử dụng cơ chế của Javascript

```
shaderProgram.vertexPositionAttribute=gl.getAttribLocation(shaderProgram,
"aVertexPosition");
gl.enableVertexAttribArray(shaderProgram.vertexPositionAttribute);

shaderProgram.textureCoordAttribute = gl.getAttribLocation(shaderProgram,
"aTextureCoord");
gl.enableVertexAttribArray(shaderProgram.textureCoordAttribute);

shaderProgram.pMatrixUniform      = gl.getUniformLocation(shaderProgram,
"uPMatrix");
shaderProgram.vMatrixUniform      = gl.getUniformLocation(shaderProgram,
"uVMatrix");
shaderProgram.wMatrixUniform      = gl.getUniformLocation(shaderProgram,
"uWMatrix");
shaderProgram.samplerUniform     = gl.getUniformLocation(shaderProgram,
"uSampler");
```

## 2. Khởi tạo buffers

Buffer chính là nơi lưu thông tin tương ứng với từng vertex. Hay nói cách khác buffer là một mảng các biến attribute của vertex (xem Hình 2-4). Số lượng buffer được sử dụng tùy thuộc vào phần code GLSL. Ví dụ như nếu hình 3D cần vẽ có màu được xác định từ một texture thì chúng ta cần tạo một buffer để lưu tọa độ texture của tất cả các vertex. Ngoài ra khi ta vẽ hình 3D theo chỉ số vertex thì ta cần thêm một buffer lưu chỉ số. Dưới đây là ví dụ khởi tạo buffer tọa độ 3D, buffer tọa độ texture và buffer chỉ số để vẽ một hình chữ vuông trong không gian ba chiều.

```
var vertexTextureCoords =[  
    0.0, 1.0,  
    1.0, 1.0,  
    1.0, 0.0,  
    0.0, 0.0  
];  
var vertexPositions = [  
    -1.0, -1.0, 0.0,  
    1.0, -1.0, 0.0,  
    1.0, 1.0, 0.0,  
    -1.0, 1.0, 0.0  
];  
var indices = [0, 1, 2, 0, 2, 3];  
  
var vertexTextureCoordBuffer = gl.createBuffer();  
gl.bindBuffer(gl.ARRAY_BUFFER, vertexTextureCoordBuffer);  
gl.bufferData(gl.ARRAY_BUFFER, new Float32Array(vertexTextureCoords),  
gl.STATIC_DRAW);  
vertexTextureCoordBuffer.itemSize = 2;  
vertexTextureCoordBuffer.numItems = vertexTextureCoords.length / 2;  
  
vertexPositionBuffer = gl.createBuffer();  
gl.bindBuffer(gl.ARRAY_BUFFER, vertexPositionBuffer);  
gl.bufferData(gl.ARRAY_BUFFER, new Float32Array(vertexPositions),  
gl.STATIC_DRAW);  
vertexPositionBuffer.itemSize = 3;  
vertexPositionBuffer.numItems = vertexPositions.length / 3;  
  
vertexIndexBuffer = gl.createBuffer();  
gl.bindBuffer(gl.ELEMENT_ARRAY_BUFFER, vertexIndexBuffer);  
gl.bufferData(gl.ELEMENT_ARRAY_BUFFER, new Uint16Array(indices),  
gl.STATIC_DRAW);  
vertexIndexBuffer.itemSize = 1;  
vertexIndexBuffer.numItems = indices.length;
```

## 3. Thực hiện thao tác vẽ

Bước 1: Chọn đối tượng program

```
gl.useProgram(shaderProgram);
```

## Bước 2: Truyền các biến attributes và uniforms.

Trong ví dụ dưới đây pMatrix, vMatrix, wMatrix , texture là các biến uniform tương ứng với uPMatrix, uVMATRIX, uWMatrix và uSampler trong code GLSL. vertexPositionBuffer và vertexTextureCoordBuffer là 2 buffer lưu tọa độ 3D và tọa độ texture của các vertex cần vẽ.

```
gl.uniformMatrix4fv(shaderProgram.pMatrixUniform, false, pMatrix);
gl.uniformMatrix4fv(shaderProgram.vMatrixUniform, false, vMatrix);
gl.uniformMatrix4fv(shaderProgram.wMatrixUniform, false, wMatrix);

gl.activeTexture(gl.TEXTURE0);
gl.bindTexture(gl.TEXTURE_2D, texture);
gl.uniform1i(shaderProgram.samplerUniform, 0);

gl.bindBuffer(gl.ARRAY_BUFFER, vertexPositionBuffer);
gl.vertexAttribPointer(shaderProgram.vertexPositionAttribute,
vertexPositionBuffer.itemSize, gl.FLOAT, false, 0, 0);

gl.bindBuffer(this.gl.ARRAY_BUFFER, vertexTextureCoordBuffer);
gl.vertexAttribPointer(shaderProgram.textureCoordAttribute,
vertexTextureCoordBuffer.itemSize, gl.FLOAT, false, 0, 0);
```

## Bước 3: Gọi hàm vẽ

Các vertex được vẽ liên tục trong mảng thì ta sử dụng hàm drawArrays (xem 2.4.7)

```
gl.drawArrays(gl.TRIANGLES, 0, vertexPositionBuffer.numItems);
```

Các vertex được vẽ chỉ số , ta sử dụng hàm drawElements. Cần tạo thêm một buffer để lưu index.

```
gl.drawElements(gl.TRIANGLES, vertexIndexBuffer.numItems, gl.UNSIGNED_SHORT
, 0);
```

### 3.2.4. Sử dụng Texture trong WebGL

#### Vấn đề

Cách sử dụng texture trong WebGL để lợp màu cho các mô hình 3D.

#### Giải pháp

Texture sử dụng để lấy màu phải có kích thước là lũy thừa của 2 ví dụ như 32 , 64 , 256, 512. Ví dụ dưới đây minh họa cách load một texture.

```

var texture;
function initTexture() {
    texture = gl.createTexture();
    texture.image = new Image();
    texture.image.onload = function () {
        handleLoadedTexture(texture)
    }
    texture.image.src = "demoTexture.gif";
}

```

Trước tiên, ta dùng hàm createTexture của đối tượng WebGL context để tạo texture. Sau đó cần xác định image của texture. Khi texture đã được load xong thì hàm handleLoadedTexture được gọi để thực hiện việc khởi tạo texture.

```

function handleLoadedTexture(texture) {
    gl.bindTexture(gl.TEXTURE_2D, texture);
    gl.pixelStorei(gl.UNPACK_FLIP_Y_WEBGL, true);
    gl.texImage2D(gl.TEXTURE_2D, 0, gl.RGBA, gl.RGBA, gl.UNSIGNED_BYTE,
    texture.image);
    gl.texParameteri(gl.TEXTURE_2D, gl.TEXTURE_MAG_FILTER, gl.NEAREST);
    gl.texParameteri(gl.TEXTURE_2D, gl.TEXTURE_MIN_FILTER, gl.NEAREST);
    gl.texParameteri(gl.TEXTURE_2D, gl.TEXTURE_WRAP_S, gl.REPEAT);
    gl.texParameteri(gl.TEXTURE_2D, gl.TEXTURE_WRAP_T, gl.REPEAT);
    gl.bindTexture(gl.TEXTURE_2D, null);
}

```

Hàm bindTexture được gọi để xác định texture ( sử dụng tương tự hàm bindBuffer x ) , những thao tác sau hàm bindTexture được thực hiện trên texture đã bind.

Trong ví dụ trên ta cần quan tâm đến hai lời gọi hàm texParameteri với tham số gl.TEXTURE\_WRAP\_S và gl.TEXTURE\_WRAP\_T. Mục đích của thao tác này là để cho biết rằng màu trên texture được lấy theo kiểu lặp lại. Tức là màu tại tọa độ (x: 1.5 , y: 0) cũng chính là màu tại tọa độ (x: 0.5 , y: 0) . Thay vì sử dụng gl.REPEAT ta có thể sử dụng gl.CLAMP\_TO\_EDGE khi đó màu tại tọa độ (x : 1.5 , y : 0) cũng là màu tại (x : 1 , y : 0)

```

gl.texParameteri(gl.TEXTURE_2D, gl.TEXTURE_WRAP_S, gl.REPEAT);
gl.texParameteri(gl.TEXTURE_2D, gl.TEXTURE_WRAP_T, gl.REPEAT);

```

Dưới đây là đoạn source code minh họa việc sử dụng texture khi vẽ hình 3D

```

gl.activeTexture(gl.TEXTURE0);

```

```
gl.bindTexture(gl.TEXTURE_2D, texture);
gl.uniform1i(shaderProgram.samplerUniform, 0);
```

Hàm activeTexture dùng để cho biết texture nào sẽ được sử dụng. WebGL có thể xử lý được tối đa 32 texture cho mỗi lần gọi hàm vẽ drawElements hoặc drawArrays, texture được đánh số từ Texture0 đến Texture31 [2]. Tiếp theo ta gọi hàm bindTexture để xác định texture được truyền xuống cho GPU xử lý thông qua hàm uniform1i. Texture được truyền như là một biến uniform (xem 2.3)

### 3.3. Tương tác của người dùng trong ứng dụng game 3D WebGL

#### Vấn đề

Cách người dùng tương tác trong game 3D bằng WebGL.

#### Giải pháp

Người dùng sẽ sử dụng bàn phím và chuột để tương tác với game 3D. Chúng ta cần bắt được các sự kiện liên quan đến phím nhấn và chuột như KeyDown , KeyUp , MouseDown, MouseUp, MouseMove.

Lớp quản lý sự kiện nhấn phím được chúng em đề xuất như sau:

```
function KeyState() {
}
KeyState.currentlyPressedKeys = [];
KeyState.handleKeyDown = function (event) {
    KeyState.currentlyPressedKeys[event.keyCode] = true;
}
KeyState.handleKeyUp = function (event) {
    KeyState.currentlyPressedKeys[event.keyCode] = false;
}
KeyState.load = function () {
    document.onkeydown = KeyState.handleKeyDown;
    document.onkeyup = KeyState.handleKeyUp;
}
```

Dưới đây là ví dụ minh họa việc kiểm tra phím ‘A’ có được nhấn hay không trong game.

```
if (KeyState.currentlyPressedKeys[65]) {
    alert('A is pressed');
```

```
}
```

Tương tự, chúng em cũng tạo lớp MouseState để quản lý các sự kiện liên quan đến chuột.

```
function MouseState() {
}
MouseState.X = null;
MouseState.Y = null;
MouseState.LEFT_BUTTON_DOWN = false;
MouseState.RIGHT_BUTTON_DOWN = false;

MouseState.MouseDown = function (e) {
    //...
}
MouseState.MouseUp = function (e) {
    //...
}
MouseState.MouseMove = function (e) {
    //...
}
MouseState.load = function (canvas) {
    canvas.onmousedown = MouseState.MouseDown;
    canvas.onmouseup = MouseState.MouseUp;
    canvas.onmousemove = MouseState.MouseMove;
}
```

### 3.4. Kiến trúc cơ bản của game 3D WebGL



#### Vấn đề

Cấu trúc cơ bản nhất của một ứng dụng game 3D trên WebGL



#### Giải pháp

Ứng dụng game 3D bằng WebGL được code bằng javascript và chạy trên browser tại client như một trang html bình thường. Cấu trúc file html chạy chương trình được nhóm em đề xuất như sau:

```
<html>
    <head>
        <!-- Các class được sử dụng -->
        <script type="text/javascript" src=".//GameModel.js"></script>

        <script type="text/javascript">
            var gl;
            var canvas;
```

```

/*Khai bao bien toan cuc*/
//...
function main() {
    /*Khoi tao canvas va gl*/
    canvas = document.getElementById("canvas");
    gl = canvas.getContext("experimental-webgl");
    gl.viewportWidth = canvas.width;
    gl.viewportHeight = canvas.height;
    gl.clearColor(0.1, 0.149, 0.237, 1.0);
    gl.enable(gl.DEPTH_TEST);

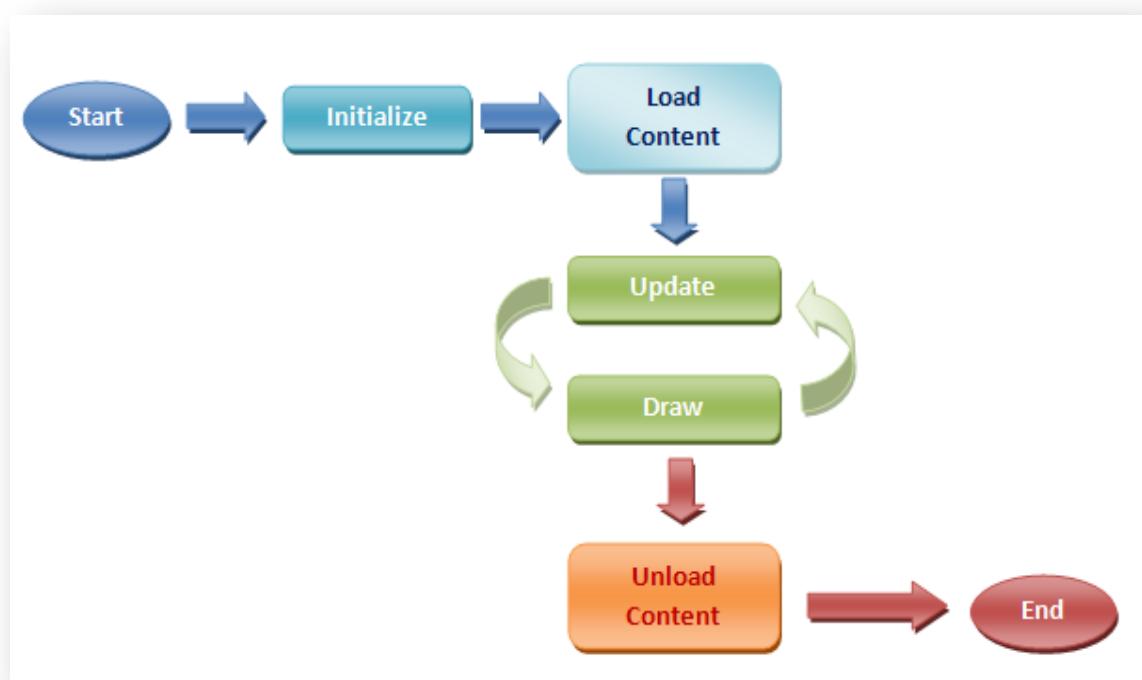
    /*Load resource*/
    Content.init(gl,"xml/resources.xml");
}
function callback() {
    /*Khoi tao cac thanh phan trong game*/
    //...
    loop();
}
function loop() {
    requestAnimFrame(loop);
    update();
    drawScene();
}
function drawScene() {
    //...
}
function update() {
    //...
}
</script>
</head>
<body onload="main();">
    <canvas id="canvas" width="800" height="600"></canvas>
</body>
</html>

```

## Các thành phần chính

- **gl:** Đối tượng WebGLRenderingContext
- **canvas :** Đối tượng canvas HTML 5
- **main:** là hàm bắt đầu chương trình để khởi tạo gl, canvas và load resources.
- **callback:** là hàm tự động gọi thực hiện sau khi đã load xong resources trong game.
- **loop:** thực hiện liên tục update và drawScene
- **update:** cập nhật logic game
- **drawScene:** vẽ các đối tượng trong game

Hàm loop là vòng lặp chính trong game nơi thực hiện liên tục các thao tác cập nhật và vẽ hình. Hình 3-4 minh họa vòng đời của một ứng dụng game.



Hình 3-4 Sơ đồ luồng xử lý của ứng dụng game

### 3.5. Kiến trúc quản lý resources game

#### Vấn đề

Một ứng dụng game WebGL sử dụng rất nhiều resources thuộc nhiều loại khác nhau như file xml, file định dạng ảnh (.png, .jpg, .bmp), file model ... Các resources này đều phải được tải về từ server theo phương pháp asynchronous tức là sẽ có một hàm được gọi thực hiện sau khi tải hoàn tất.

Ví dụ dưới đây là đoạn code dùng để load texture, hàm *handleLoadedTexture* khởi tạo texture sẽ được gọi sau khi đã load xong texture.

```
function loadTexture() {  
    var texture = gl.createTexture();  
    texture.image = new Image();  
    texture.image.onload = function() {  
        handleLoadedTexture(texture)  
    }  
    texture.image.src = "nehe.gif";  
}
```

```

    }
    function handleLoadedTexture(texture) {
        gl.bindTexture(gl.TEXTURE_2D, texture);
        gl.pixelStorei(gl.UNPACK_FLIP_Y_WEBGL, true);
        gl.texImage2D(gl.TEXTURE_2D, 0, gl.RGBA, gl.RGBA, gl.UNSIGNED_BYTE,
        texture.image);
        gl.texParameteri(gl.TEXTURE_2D, gl.TEXTURE_MAG_FILTER, gl.NEAREST);
        gl.texParameteri(gl.TEXTURE_2D, gl.TEXTURE_MIN_FILTER, gl.NEAREST);
        gl.bindTexture(gl.TEXTURE_2D, null);
    }
}

```

Vấn đề đặt ra là trong chương trình cần phải sử dụng nhiều resource tuy nhiên có thể resource đó chưa load xong. Có thể giải quyết bằng cách sử dụng các lệnh if kiểm tra xem resource đã được load xong hay chưa trước khi sử dụng nhưng cách này làm source code trở nên phức tạp và dài dòng.



## Giải pháp

Nhóm em đã tạo ra lớp Content để quản lý các resource trong game. Khi cần sử dụng một resource ta chỉ cần gọi làm load với tham số là id của resource. Lớp Content hỗ trợ load xml , model , effect , texture.

```

var xmlDoc = Content.load["xml"](xmlID);
var texture = Content.load["texture"](textureID);
var dataModel = Content.load["model"](modelID);
var effectCode = Content.load["effect"](effectID);

```

Chúng ta cần khởi tạo lớp Content ở đầu chương trình.

```
Content.init("xml/resources.xml");
```

Hàm callback được gọi khi đã load xong tất cả các resources

```

function callback() {
}

```

Cấu trúc file resources.xml quản lý resource được nhóm em đề xuất như sau:

```

<!-- model/character -->
<model url="model/character/hero/lichkingbody.json" id="lichkingbody"/>
<model url="model/character/monster/pitlord/pitlord.json" id="pitlord"/>
<!-- effect -->

```

```

<effect url="effect/basiceffect.json" id="basiceffect"/>
<effect url="effect/terraineffect.json" id="terraineffect"/>
<!-- xml -->
<xml url="xml/maps/mapvillage.xml" id="mapvillage"/>
<xml url="xml/maps/mapadventure.xml" id="mapadventure"/>
<!-- texture/terrain -->
<texture url="texture/terrain/grass.png" id="grass"/>
<texture url="texture/terrain/snow.jpg" id="snow"/>
<texture url="texture/terrain/water.jpg" id="water"/>

```

### 3.6. Kiến trúc liên quan đến lập trình điều khiển GPU - Effect

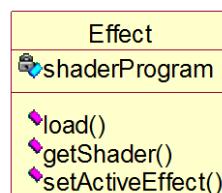
#### Vấn đề

Việc thực hiện các thao tác đồ họa 3D bằng WebGL đòi hỏi phải hiểu rõ phương pháp lập trình GPU bằng GLSL và cách tương tác giữa phần source code javascript WebGL với code GLSL do đó chúng ta cần một phương pháp hiệu quả để đơn giản hóa việc lập trình WebGL. Ngoài ra, trong ứng dụng game thực tế chúng ta thường có nhu cầu vẽ các đối tượng 3D với các hiệu ứng hoàn toàn khác nhau điều này đặt ra vấn đề làm sao để quản lý hiệu quả các hiệu ứng trong game.

#### Giải pháp

Nhóm em tạo ra lớp Effect để quản lý các hiệu ứng trong game. Lớp Effect sẽ có nhiệm vụ load và xử lý code GLSL đồng thời đóng gói các thành phần cần thiết cho việc vẽ 3D. Các thành phần trong lớp Effect gồm có:

- **shaderProgram**: Đối tượng shaderProgram
- **load**: Phương thức dùng để load code GLSL và khởi tạo Effect.
- **getShader**: Phương thức được dùng nội bộ trong lớp Effect để tạo vertex shader và fragment shader.
- **setActiveEffect**: Phương thức dùng để báo cho hệ thống biết là sẽ sử dụng effect này để thực hiện thao tác vẽ hình 3D.



Hình 3-5 Lớp Effect

Dưới đây là một ví dụ về cách sử dụng lớp Effect trong game

```
function GameModel() {
}
GameModel.prototype.load = function(modelID, textureID, effectID) {
    //Other Initialization
    //...

    //Load effect
    this.effect = new Effect();
    this.effect.load(effectID);
}
GameModel.prototype.draw = function(camera, wMatrix) {
    //Set active effect
    this.effect.setActiveEffect();
    this.setEffectParams();

    //Draw operations
    //...
}
GameModel.prototype.setEffectParams = function() {
    //Set effect parameters
    //...
}
```

Effect được load từ file code GLSL được xác định bởi effectID (xem phần 3.5).

Phần code GLSL bao gồm Vertex Shader và Fragment Shader. Cấu trúc của file code GLSL được nhóm em định nghĩa như sau:

```
//Begin Vertex Shader
attribute vec3 aVertexPosition;
attribute vec2 aTextureCoord;
uniform mat4 uVMatrix;
uniform mat4 uPMatrix;
uniform mat4 uWMatrix;
varying vec2 vTextureCoord;
void main(void) {
    gl_Position=uPMatrix*uVMatrix*uWMatrix*vec4(aVertexPosition, 1.0);
    vTextureCoord = aTextureCoord;
}
//Begin Fragment Shader
precision mediump float;
varying vec2 vTextureCoord;
uniform sampler2D uSampler;
void main(void) {
    gl_FragColor=texture2D(uSampler, vec2(vTextureCoord.s, vTextureCoord.t));
}
```

### 3.7. Vẽ đa giác trong lập trình 3D với WebGL



Vấn đề

Các đối tượng từ đơn giản nhất cho đến phức tạp nhất trong không gian 3D đều được cấu tạo từ một tập hợp những tam giác đơn vị nhỏ (Primitive). Các primitives này lại được tạo bởi một bộ 3 vertices. Khi vẽ một đối tượng 3D thực chất dữ liệu được truyền cho GPU để vẽ đó là một mảng các vertices định nghĩa các primitives mà kết hợp với nhau tạo thành một đối tượng 3D hoàn chỉnh. Phần này nói về như thế nào để vẽ được một mảng đỉnh.



## Giải pháp

Dưới đây là một ví dụ minh họa cách vẽ một đa giác. Trong ví dụ này chúng em định nghĩa lớp Rectangle để vẽ một hình chữ nhật có các đỉnh lần lượt là  $(-1,-1,0)$ ,  $(1,-1,0)$ ,  $(1,1,0)$ ,  $(-1,1,0)$ .

```
function Rectangle() {
}
Rectangle.prototype.load = function(){
    var vertexPositions = [
        -1.0, -1.0, 0.0,
        1.0, -1.0, 0.0,
        1.0, 1.0, 0.0,
        -1.0, 1.0, 0.0
    ];
    var indices = [0, 1, 2, 0, 2, 3];

    this.vertexPositionBuffer = gl.createBuffer();
    gl.bindBuffer(gl.ARRAY_BUFFER, this.vertexPositionBuffer);
    gl.bufferData(gl.ARRAY_BUFFER, new Float32Array(vertexPositions),
gl.STATIC_DRAW);
    this.vertexPositionBuffer.itemSize = 3;
    this.vertexPositionBuffer.numItems = vertexPositions.length / 3;

    this.vertexIndexBuffer = gl.createBuffer();
    gl.bindBuffer(gl.ELEMENT_ARRAY_BUFFER, this.vertexIndexBuffer);
    gl.bufferData(gl.ELEMENT_ARRAY_BUFFER, new Uint16Array(indices),
gl.STATIC_DRAW);
    this.vertexIndexBuffer.itemSize = 1;
    this.vertexIndexBuffer.numItems = indices.length;

    this.effect = new Effect();
    this.effect.load("basiceffect");
}

Rectangle.prototype.draw = function(camera){
    this.effect.setActiveEffect();
    this.setEffectParams(camera.pMatrix,camera.vMatrix);
    gl.bindBuffer(gl.ELEMENT_ARRAY_BUFFER, this.vertexIndexBuffer);
    gl.drawElements(gl.TRIANGLES, this.vertexIndexBuffer.numItems,gl.UNSIGN
ED_SHORT, 0);
}

Rectangle.prototype.setEffectParams = function(pMatrix,vMatrix) {
    var shaderProgram = this.effect.shaderProgram;
```

```

        if(shaderProgram.isReady == null) {
            shaderProgram.vertexPositionAttribute=gl.getAttribLocation(shaderProgram, "aVertexPosition");

            gl.enableVertexAttribArray(shaderProgram.vertexPositionAttribute);

            shaderProgram.pMatrixUniform = gl.getUniformLocation(shaderProgram, "uPMatrix");
            shaderProgram.vMatrixUniform = gl.getUniformLocation(shaderProgram, "uVMatrix");
            shaderProgram.isReady = true;
        }
        gl.uniformMatrix4fv(shaderProgram.pMatrixUniform, false, pMatrix);
        gl.uniformMatrix4fv(shaderProgram.vMatrixUniform, false, vMatrix);

        gl.bindBuffer(gl.ARRAY_BUFFER, this.vertexPositionBuffer);
        gl.vertexAttribPointer(shaderProgram.vertexPositionAttribute, this.vertexPositionBuffer.itemSize, gl.FLOAT, false, 0, 0);
    }
}

```

Các thành phần trong lớp Rectangle gồm có:

- **vertexPositionBuffer:** Đôi tượng buffer lưu vị trí các vertex
- **vertexIndexBuffer:** Đôi tượng buffer lưu chỉ số khi vẽ các vertex
- **effect:** Đôi tượng effect
- **load:** Hàm khởi tạo Rectangle, bao gồm các thao tác
  - Tạo mảng vertexPosition
  - Tạo mảng indices
  - Khởi tạo vertexPositionBuffer từ mảng vertexPosition
  - Khởi tạo vertexIndexBuffer từ mảng indices
  - Load effect với file code GLSL có id là “basiceffect”
- **draw:** Hàm thực hiện thao tác vẽ, bao gồm các thao tác
  - setActiveEffect
  - setEffectParams
  - bind vertexIndexBuffer
  - drawElements : với tham số gl.Triangles
- **setEffectParams:** Hàm thực hiện truyền các thông tin xuống GPU xử lý.
  - Xác định vị trí của các biến được sử dụng trong code GLSL như “*aVertexPosition*”, “*uPMatrix*” và “*uVMatrix*”.
  - Sau khi đã có vị trí các biến, bắt đầu truyền giá trị thông qua các hàm *gl.uniformMatrix4fv*, *gl.vertexAttribPointer*

Dưới đây là nội dung file code GLSL được sử dụng cho lớp Rectangle

```

//Begin vertex shader
attribute vec3 aVertexPosition;
uniform mat4 uVMMatrix;
uniform mat4 uPMatrix;
void main(void) {
    gl_Position = uPMatrix*uVMMatrix*vec4(aVertexPosition, 1.0);
}
//Begin fragment shader
precision mediump float;
void main(void) {
    gl_FragColor = vec4(1.0, 1.0, 1.0, 1.0);
}

```

Cách sử dụng lớp Rectangle được trình bày trong đoạn code dưới đây.

```

var rect;
function callback() {
    /*Khoi tao*/
    /**
     rect = new Rectangle();
     rect.load();
     loop();
}
function drawScene() {
    /**
     rect.draw(camera);
}

```

Kết quả là chúng ta có được một hình chữ nhật màu trắng có tọa độ :

$(-1.0, -1.0, 0.0)$  ,  $(1.0, -1.0, 0.0)$  ,  $(1.0, 1.0, 0.0)$  ,  $(-1.0, 1.0, 0.0)$

### 3.8. Kết luận

Thông qua chương này, nhóm đã giới thiệu những vấn đề cơ bản nhất liên quan đến đồ họa sử dụng WebGL. Những vấn đề này sẽ là nền tảng và yêu cầu cần thiết để có thể nắm bắt được những vấn đề nâng cao như sử dụng mô hình 3D và cách thực hiện nhiều loại hiệu ứng 3D khác nhau.

## Chương 4

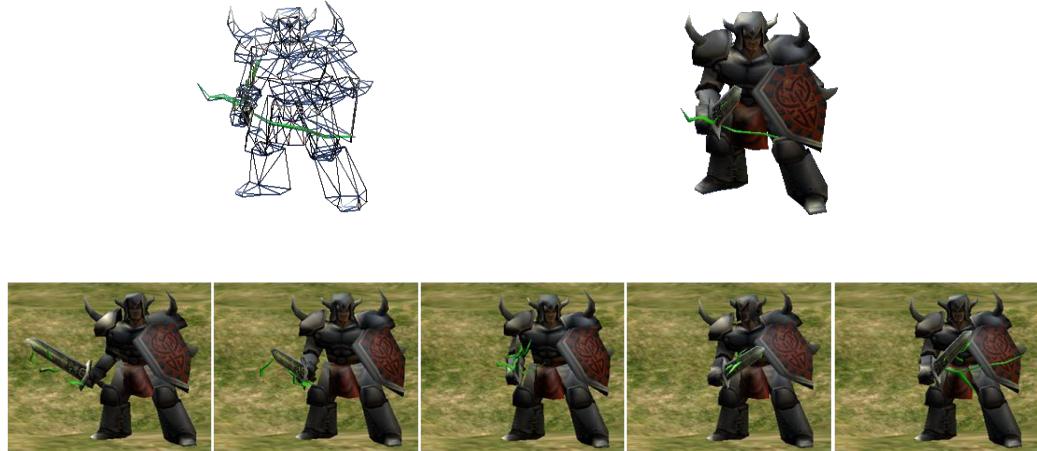
# Các vấn đề liên quan đến mô hình 3D

 Nội dung chương này trình bày các vấn đề về tạo lập, quản lý và xử lý những hành động của nhân vật trong không gian ba chiều trên môi trường WebGL.

### 4.1. Giới thiệu về đặc trưng mô hình 3D

Không gian 3 chiều được xây dựng gồm những khối vật thể đặt trong thế giới 3D, những khối vật thể này được tạo bằng việc vẽ các tam giác đơn vị (primitives) tạo nên bề mặt của khối vật thể. Nhưng nếu phải lập trình để điều khiển việc vẽ tập hợp các tam giác đơn vị này thì việc tạo nên một khối vật thể có hình dạng đẹp là một vấn đề rất khó trong thuật toán cũng như đòi hỏi nhiều chi phí trong việc tính toán. Để đơn giản hóa việc tạo dựng các khối vật thể 3D, một giải pháp đặt ra là sử dụng các mô hình 3D được thiết kế sẵn bằng những công cụ chuyên dụng cho việc thiết kế khối vật thể ở bên ngoài ứng dụng game. Từ đó chúng tỏ rằng một trong những yếu tố rất quan trọng và quyết định sự thành bại về đồ họa trong game 3D đó chính là mô hình 3 chiều mà trong chương này sẽ tập trung vào các mô hình 3D cho nhân vật.

Về mặt cơ bản các mô hình 3D bao gồm một mesh các tam giác đơn vị cùng với các thông tin thiết lập cho việc phủ texture trên bề mặt của mô hình. Nếu dùng ở đó thì thứ có được từ một mô hình 3D chỉ là một khối vật thể nhất định. Nhưng thực tế một nhân vật tồn tại trong không gian 3 chiều không hề đứng yên mà sẽ có những cử động liên tục. Để đáp ứng cho nhu cầu này các mô hình được thiết kế cho nhân vật nói riêng hay vật thể sống nói chung sẽ cung cấp những phép biến đổi bên trong mô hình để các thành phần của mô hình có thể di chuyển theo một quy luật cho trước tạo nên những cử động của nhân vật. Và một nhân vật sẽ có nhiều loại hành động khác nhau tùy thuộc vào nhu cầu của người tạo lập game. Ví dụ như: hành động đứng yên tại chỗ (stand), hành động di chuyển (walk), hành động tấn công (attack) ...



**Hình 4-1 Mô hình nhân vật 3 chiều**

Hơn thế nữa ở bước phát triển cao hơn, với một số loại mô hình nhân vật, thành phần trong nó không đơn thuần chỉ bao gồm duy nhất một mesh các tam giác đơn vị như mô hình bình thường mà mỗi mô hình nhân vật bao gồm rất nhiều những khối mesh khác nhau được gắn kết theo một khung xương (bone) nhất định. Khung xương này giúp dễ dàng xác định vị trí tạo độ mốc của các phần trên cơ thể nhân vật mà từ đó có thể dễ dàng điều khiển, lắp thêm vũ khí, đồ đạc thậm chí tạo các hiệu ứng phép thuật đẹp mắt trên các phần cơ thể này. Diễn hình một loại mô hình vừa hỗ trợ animation vừa có khung xương đó là loại mô hình Quake3 (\*.md3).

Vậy nội dung của chương này sẽ trình bày các vấn đề để tạo lập (load mô hình từ file mô hình và create đối tượng mô hình nhân vật), quản lý (điều khiển trong suốt quá trình tồn tại của mô hình), và xử lý những hành động (thay đổi animation của mô hình nhân vật) được thực hiện trên môi trường WebGL.

#### Các vấn đề được đề cập bao gồm:

- Load mô hình 3D
- Vẽ mô hình 3D
- Sử dụng mô hình 3D trong code
- Thay đổi tỷ lệ mô hình 3D
- Animation của mô hình 3D
- Kiến trúc sử dụng mô hình 3D trong ứng dụng game

## 4.2. Load mô hình 3D

### Vấn đề

Nội dung phần này trình bày cách load một mô hình 3D trong WebGL

### Giải pháp

Dưới đây là phần định nghĩa lớp GameModel do chúng em đề xuất. Về cơ bản để vẽ được một mô hình 3D chúng ta cần file mô hình, texture và code GLSL. Phương thức load của GamemModel nhận các tham số là modelID, textureID, effectID (xem phần 3.5)

```
function GameModel() {
}
GameModel.prototype.load = function(modelID, textureID, effectID) {

    //Load model
    var data = Content.load["model"] (modelID);
    this.vertexTextureCoordBuffer = gl.createBuffer();
    gl.bindBuffer(gl.ARRAY_BUFFER, this.vertexTextureCoordBuffer);
    gl.bufferData(gl.ARRAY_BUFFER, new Float32Array(data.vertexTextureCoords),
    gl.STATIC_DRAW);
    this.vertexTextureCoordBuffer.itemSize = 2;
    this.vertexTextureCoordBuffer.numItems = data.vertexTextureCoords.length/2;

    this.vertexPositionBuffer = gl.createBuffer();
    gl.bindBuffer(gl.ARRAY_BUFFER, this.vertexPositionBuffer);
    gl.bufferData(gl.ARRAY_BUFFER, new Float32Array(data.vertexPositions),
    gl.STATIC_DRAW);
    this.vertexPositionBuffer.itemSize = 3;
    this.vertexPositionBuffer.numItems = data.vertexPositions.length / 3;

    this.vertexIndexBuffer = gl.createBuffer();
    gl.bindBuffer(gl.ELEMENT_ARRAY_BUFFER, this.vertexIndexBuffer);
    gl.bufferData(gl.ELEMENT_ARRAY_BUFFER, new Uint16Array(data.indices),
    gl.STATIC_DRAW);
    this.vertexIndexBuffer.itemSize = 1;
    this.vertexIndexBuffer.numItems = data.indices.length;

    //Load texture
    this.texture = Content.load["texture"] (textureID);

    //Load effect
    this.effect = new Effect();
    this.effect.load(effectID);

}
```

Các bước để load một model 3D :

- **Bước 01** : Load file JSON của model
- **Bước 02** : Khởi tạo các buffers
- **Bước 03** : Load texture
- **Bước 04** : Load effect

## (1) Load JSON

Định dạng của một file JSON mô tả mô hình 3D cơ bản sẽ bao gồm 3 mảng vertexPositions, vertexTextureCoords, indices (xem Hình 4-2). Nếu chúng ta cần giải quyết thêm các vấn đề khác như animation và boundingbox của mô hình thì file JSON sẽ chứa thêm một số thông tin khác.

- **vertexPositions**: Mảng lưu tọa độ tất cả các đỉnh trong model
- **vertexTextureCoords**: Mảng lưu tọa độ texture của tất cả các đỉnh, mỗi một đỉnh cần có một tọa độ texture để xác định màu cho đỉnh đó.
- **indices**: Mảng lưu chỉ số của thao tác vẽ, nói cách khác chính là thứ tự vẽ các đỉnh để tạo thành một mô hình 3D.

```
{  
    "vertexPositions" : [0.000000,0.002000,0.002000,0.000000,-0.002000,0.002000,0.000000  
    "vertexTextureCoords" : [0.629500,0.503900,0.629500,0.503900,0.629500,0.503900,0.62  
    "indices" : [0,1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16,17,18,19,20,21,22,23,24,25,26  
}
```

Hình 4-2 Mô hình JSON

## (2) Khởi tạo buffers

Chúng ta cần các đối tượng buffer để lưu trữ **vertexPositions**, **vertexTextureCoords**, **indices**. Dưới đây là các bước để khởi tạo buffers

- Đầu tiên là ta tạo buffer qua hàm **gl.createBuffer()**
- Sau đó gọi hàm **gl.bindBuffer()** để xác định buffer được thao tác
- Truyền dữ liệu mảng vào buffer thông qua hàm **gl.bufferData()**.
- Thuộc tính **itemSize** được thêm vào cho đối tượng buffer thông qua cơ chế trong Javascript nó cho biết kích thước của mỗi phần tử trong buffer, chẳng hạn đối

với ***vertexPositionBuffer***, ***itemSize*** bằng 3 là do tọa độ mỗi đỉnh gồm 3 thành phần. Tương tự ***vertexTextureCoordBuffer*** có ***itemSize*** là 2.

- Thuộc tính ***numItems*** cũng được thêm vào buffer qua cơ chế trong Javascript, nó cho biết số lượng các phần tử có trong mỗi buffer.

### (3) Load texture

Thông qua lời gọi hàm ***Content.load["texture"](*textureID*)*** ta load được một texture hoàn chỉnh và có thể sử dụng được ngay (xem 3.5).

### (4) Load effect

Tham khảo phần 3.6.

## 4.3. Vẽ mô hình 3D

Để vẽ được mô hình 3D ta cần sử dụng projection matrix và view matrix của camera (xem 5.1). Ngoài ra cần phải có world matrix để xác định các phép biến đổi đối với model bao gồm translation, scale, rotation. GameModel sẽ thực hiện thao tác vẽ với bất kỳ world matrix nào được đưa cho nó. Việc lưu giữ lại thông số của các phép biến đổi và world matrix sẽ do một đối tượng logic khác quản lý ví dụ như Building , Character (xem Hình 4-6)

```
GameModel.prototype.draw = function(camera,wMatrix){  
  
    this.effect.setActiveEffect();  
    this.setEffectParams(camera.pMatrix,camera.vMatrix,wMatrix);  
    gl.bindBuffer(gl.ELEMENT_ARRAY_BUFFER, this.vertexIndexBuffer);  
    gl.drawElements(gl.TRIANGLES,this.vertexIndexBuffer.numItems,gl.UNSIGNED_  
    SHORT, 0);  
  
}  
  
GameModel.prototype.setEffectParams = function(pMatrix,vMatrix,wMatrix){  
  
    var shaderProgram = this.effect.shaderProgram;  
    if(shaderProgram.isReady == null){  
        shaderProgram.vertexPositionAttribute =  
            gl.getAttribLocation(shaderProgram, "aVertexPosition");  
  
        gl.enableVertexAttribArray(shaderProgram.vertexPositionAttribute);  
  
        shaderProgram.textureCoordAttribute =  
            gl.getAttribLocation(shaderProgram, "aTextureCoord");  
        gl.enableVertexAttribArray(shaderProgram.textureCoordAttribute);  
    }  
}
```

```

        shaderProgram.pMatrixUniform = gl.getUniformLocation(shaderProgram,
"uPMatrix");
        shaderProgram.vMatrixUniform = gl.getUniformLocation(shaderProgram,
"uVMatrix");
        shaderProgram.wMatrixUniform = gl.getUniformLocation(shaderProgram,
"uWMatrix");
        shaderProgram.samplerUniform = gl.getUniformLocation(shaderProgram,
"uSampler");

        shaderProgram.isReady = true;
    }

gl.uniformMatrix4fv(shaderProgram.pMatrixUniform, false, pMatrix);
gl.uniformMatrix4fv(shaderProgram.vMatrixUniform, false, vMatrix);
gl.uniformMatrix4fv(shaderProgram.wMatrixUniform, false, wMatrix);

gl.activeTexture(gl.TEXTURE0);
gl.bindTexture(gl.TEXTURE_2D, this.texture);
gl.uniform1i(shaderProgram.samplerUniform, 0);

gl.bindBuffer(gl.ARRAY_BUFFER, this.vertexPositionBuffer);
gl.vertexAttribPointer(shaderProgram.vertexPositionAttribute,
this.vertexPositionBuffer.itemSize, gl.FLOAT, false, 0, 0);

gl.bindBuffer(gl.ARRAY_BUFFER, this.vertexTextureCoordBuffer);
gl.vertexAttribPointer(shaderProgram.textureCoordAttribute,
this.vertexTextureCoordBuffer.itemSize, gl.FLOAT, false, 0, 0);

}

```

Các bước để vẽ một model 3D

- **Bước 01 :** Chọn effect thông qua hàm  *setActiveEffect*
- **Bước 02 :** Gọi hàm  *setEffectParams* để truyền các tham số xuống GPU
- **Bước 03 :** Bind  *vertexIndexBuffer* và gọi  *drawElements*

### (1) Chọn effect

Tham khảo phần 3.6. Dưới đây là phần code GLSL sẽ được sử dụng trong GameModel.

```

//Begin vertex shader
attribute vec3 aVertexPosition;
attribute vec2 aTextureCoord;
uniform mat4 uVMatrix;
uniform mat4 uPMatrix;
uniform mat4 uWMatrix;
varying vec2 vTextureCoord;
void main(void) {

```

```

gl_Position = uPMatrix*uVMatrix*uWMatrix*vec4(aVertexPosition, 1.0);
vTextureCoord = aTextureCoord;
}
//Begin fragment shader
precision mediump float;
varying vec2 vTextureCoord;
uniform sampler2D uSampler;
void main(void) {
    gl_FragColor=texture2D(uSampler, vec2(vTextureCoord.s,vTextureCoord.t));
}

```

## (2) setEffectParams

Mục đích của hàm ***setEffectParams*** là để truyền các thông số cần thiết cho quá trình vẽ hình 3D. Trước khi bắt đầu truyền ta cần phải xác định địa chỉ các biến của phần code GLSL. Với biến kiểu attribute ta sử dụng hàm ***gl.getAttributeLocation***. Tương tự, biến kiểu uniform được lấy địa chỉ bằng ***gl.getUniformLocation***. Địa chỉ các biến được lưu tạm thành những property trong đối tượng ***shaderProgram***. Đối với biến attribute, sau khi có được địa chỉ ta cần gọi hàm ***gl.enableVertexAttribArray***.

Chúng ta sử dụng hàm ***gl.vertexAttribPointer*** để thực hiện truyền các biến attribute và dùng các hàm có dạng ***gl.uniform*** để truyền các biến uniform, ví dụ như ***gl.uniformMatrix4fv*** để truyền các ma trận , ***gl.uniform1f*** để truyền một số thực

## (3) Bind index buffer và gọi hàm drawElements

Ta gọi hàm ***gl.bindBuffer*** để xác định buffer hiện tại là ***vertexIndexBuffer***, sau đó gọi hàm ***gl.drawElements*** để thực hiện vẽ.



**Hình 4-3 Kết quả vẽ một mô hình 3D**

#### **4.4. Sử dụng mô hình 3D trong code**

Dưới đây là đoạn code minh họa cách sử dụng mô hình 3D đơn giản nhất trong ứng dụng game. Chúng ta load một mô hình 3D và khởi tạo một ma trận world (wMatrix), sau đó thực hiện thao tác vẽ mô hình với ma trận world đã có.

```
var model;
var wMatrix;
function callback() {
    /*Khoi tao*/
    //...
    model = new GameModel();
    model.load("townhall","townhall","basiceffect");
    wMatrix = mat4.create();
    mat4.identity(wMatrix);
    loop();
}
function drawScene() {
    //...
    model.draw(camera,wMatrix);
}
```

#### **4.5. Thay đổi tỷ lệ mô hình**

**Vấn đề**

Sau khi load được mô hình lên ứng dụng game, kích thước gốc của mô hình khi thiết kế có thể quá lớn so với điều kiện ngoại cảnh thực tế nơi mô hình cần được vẽ. Khi đó cần phải thay đổi tỷ lệ kích thước (scale) của mô hình để thu nhỏ hay phóng lớn mô hình cho phù hợp với ngoại cảnh. Việc thay đổi kích thước này được thực hiện trên cả 3 chiều x, y, z của không gian. Để giữ nguyên hình dạng ban đầu của mô hình có thể thực hiện scale với cùng một tỷ lệ trên 3 chiều của không gian ngược lại có thể làm biến dạng mô hình như kéo dãn hay co ép lại theo các chiều khác nhau.



### Giải pháp

Nhắc lại các ma trận cần thiết để vẽ một đối tượng trên thế giới 3D đó là world matrix, view matrix và projection matrix. Trong đó view matrix và projection matrix thiết lập các thông tin về không gian nhìn của camera, còn world matrix là ma trận xác định việc đối tượng sẽ được đặt vào thế giới 3D như thế nào. World matrix có thể được tổng hợp từ các phép biến đổi bao gồm translation, scale, rotation.

Để thay đổi kích thước của mô hình về cơ bản chúng ta cần nhân thêm ma trận scale vào ma trận world.

```
var wMatrix = mat4.create();
mat4.identity(wMatrix);
mat4.scale(wMatrix, [scalex, scaley, scalez]);
```

Trong ví dụ trên scalex, scaley, scalez lần lượt là hệ số scale theo các trục x , y, z, thông thường thì chúng ta sử dụng cùng một giá trị cho scalex , scaley, scalez.

### ❖ Kết luận

Việc thay đổi tỷ lệ của mô hình đối tượng theo cách trên là một cách dễ dàng và đơn giản để điều khiển việc vẽ một mô hình nhân vật với kích thước được quy định trước sao cho phù hợp với ngoại cảnh hiện tại đang được vẽ trong thế giới 3 chiều. Đây là một kỹ thuật cơ bản nhưng rất quan trọng trong việc sử dụng mô hình trong game 3D.

## 4.6. Animation của Mô hình 3D

Như vậy là ta đã load được model vào không gian 3D, có thể thay đổi vị trí và góc quay của model. Nhưng những model này chưa có cử chỉ, hành động.

Có nhiều dạng hành động khác nhau như : chế độ đứng yên (stand), di chuyển (walk), chạy (run), tấn công (attack), ... Thật ra, Animation là một mảng các Frame (tức vertexPosition) ghép lại với nhau một cách hài hòa và hợp lý, tạo ra hành động cho nhân vật. Một model muốn Animation mượt, hành động đẹp mắt thì cần có một mảng chứa số lượng lớn Frame. Thường thường thì 60 Frame cho một Action là di chuyển cũng mượt 80%.

Mô hình animation về cơ bản là một tập hợp các mô hình 3D tĩnh. Điểm khác nhau giữa file JSON của mô hình có animation và mô hình không có animation đó là thay vì chỉ lưu một mảng *vertexPositions* chúng ta sẽ lưu nhiều mảng *vertexPositions* tương ứng nhiều frame khác nhau trong animation.

```
{  
    "vertexPositions" : [0.000000,0.002000,0.002000,0.000000,-0.002000,0.002000,0.000000  
    "vertexTextureCoords" : [0.629500,0.503900,0.629500,0.503900,0.629500,0.503900,0.62  
    "indices" : [0,1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16,17,18,19,20,21,22,23,24,25,26  
}
```

Hình 4-4 Mô hình không có animation

```
{  
    "frames_walk" : [[-2.348614,149.595642,58.369827,2.604914,148.043823,61.614227,1.121759,149.835205,62.920380,5.1125,  
    "vertexTextureCoords" : [0.629500,0.503900,0.629500,0.503900,0.629500,0.503900,0.629500,0.503900,0.629500,0.503900,  
    "indices" : [0,1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16,17,18,19,20,21,22,23,24,25,26,27,28,29,30,31,32,33,34,35,36,37  
}
```

Hình 4-5 Mô hình có animation

Trong Hình 4-5 ta thấy có thêm *frames\_walk* đó là một mảng bao gồm nhiều mảng *vertexPositions*. Lớp AnimationModel do nhóm em cài đặt như sau :

```
function AnimationModel () {  
}  
AnimationModel.inheritsFrom(GameModel);
```

```

AnimationModel.prototype.load = function(modelID, textureID, effectID) {
    /* Khoi tao tuong tu GameModel */
    //...

    this.animation = {};
    this.animation.walk    = data.frames_walk;
    this.animation.stand   = data.frames_stand;
    this.animation.attack  = data.frames_attack;

    this.currentFrameIndex = 0;
    this.currentAnimation  = "walk";
    this.elapsed = 0;
}
AnimationModel.prototype.updateAnimation = function (elapsed) {

this.elapsed += elapsed;
if(this.elapsed >= 40){
    this.currentFrameIndex++;
    this.currentFrameIndex=this.currentFrameIndex%this.animation[this.currentAnimation].length;

    gl.bindBuffer(gl.ARRAY_BUFFER, this.vertexPositionBuffer);
    gl.bufferData(gl.ARRAY_BUFFER, new
Float32Array(this.animation[this.currentAnimation] [this.currentFrameIndex
]), gl.STATIC_DRAW);

    this.elapsed = 0;
}
}

AnimationModel.prototype.playAnimation = function(animation){
    this.currentAnimation = animation;
}

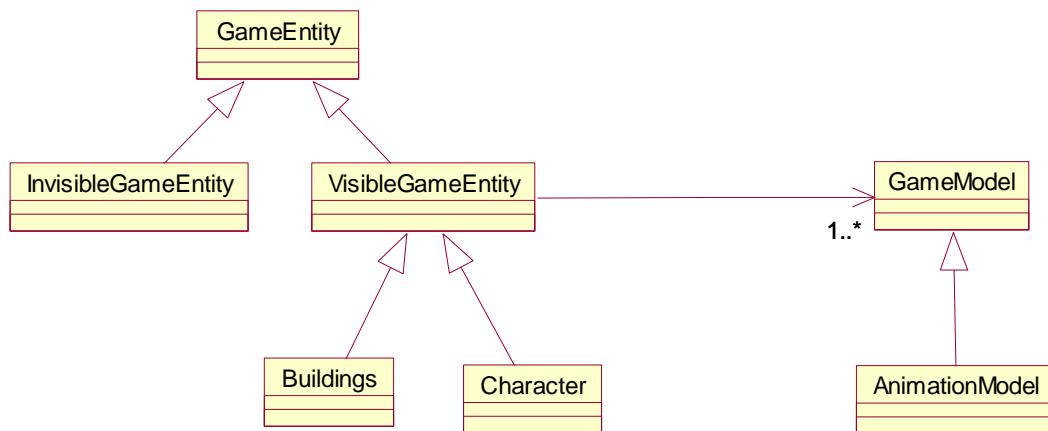
```

Ngoài phần khởi tạo tương tự lớp GameModel, trong lớp AnimationModel chúng ta cần có một thuộc tính lưu lại tất cả những loại animation. Trong đoạn code bên trên, mỗi mô hình animation sẽ có 3 loại animation là ***walk***, ***stand*** , ***attack***. Mỗi loại animation là một mảng các frame liên tục. ***currentFrameIndex*** được dùng để xác định chỉ số frame hiện tại. ***currentAnimation*** xác định loại animation hiện tại. Ngoài ra chúng ta còn cần một biến ***elapsed*** để lưu thời gian.

Phương thức updateAnimation có nhiệm vụ xác định frame tiếp theo của animation hiện tại. Sau khi xác định được frame, chúng ta có thể truyền chính xác mảng tọa độ định vào ***this.vertexPositionBuffer*** bằng hàm ***gl.bufferData***.

`this.animation[this.currentAnimation][this.currentFrameIndex]` chính là mảng tọa độ vertex hiện tại cần phải vẽ. Phương thức `playAnimation` được dùng để thay đổi loại animation của mô hình. Thời gian thay đổi chỉ số frame vào khoảng 40 milli second (tương đương 25 frames trong một giây)

#### 4.7. Kiến trúc sử dụng mô hình 3D trong ứng dụng.



Hình 4-6 Kiến trúc sử dụng mô hình 3D

Mô hình 3D trong ứng dụng được đại diện bằng lớp GameModel. Một đối tượng GameModel có nhiệm vụ vẽ ra màn hình với các thông số truyền vào như world matrix, projection matrix, view matrix. Tất cả các thông tin liên quan đến logic game như vị trí, góc xoay, tỷ lệ scale của mô hình được quản lý trong các lớp kế thừa từ VisibleGameEntity như Building , Character. Dưới đây là đoạn code định nghĩa lớp Character.

```

function Character() {
}
Character.inheritsFrom(VisibleGameEntity);
Character.prototype.load = function(){
    /* Khoi tao */
    //...
    this.model = new AnimationModel();
    this.model.load(modelID,textureID,effectID);
}
Character.prototype.draw = function(camera){
    this.model.draw(camera,this.getWorldMatrix());
}
VisibleGameEntity.prototype.getWorldMatrix = function(){
    var wMatrix = mat4.create();
    mat4.identity(wMatrix);
}
  
```

```
    mat4.translate(wMatrix, [this.translate.x, this.translate.y, this.translate.z]);
    mat4.rotateY(wMatrix, Util.degToRad(this.rotate.y));
    mat4.rotateX(wMatrix, Util.degToRad(this.rotate.x));
    mat4.rotateZ(wMatrix, Util.degToRad(this.rotate.z));
    mat4.scale(wMatrix, [this.scale.x, this.scale.y, this.scale.z]);
    return wMatrix;
}
```

Khi vẽ một Character thực chất là chúng ta gọi hàm draw của AnimationModel với những thông số của Character. Với phương pháp này chúng ta hoàn toàn có thể làm được việc là một AnimationModel được sử dụng bởi nhiều Character giúp nâng cao hiệu xuất của game.

## 4.8. Kết luận

Nội dung của chương này trình bày các vấn đề liên quan đến việc tạo lập mô hình 3D từ file JSON và cách thực hiện hiệu ứng animation trên mô hình 3D, đây là một phương pháp quan trọng được sử dụng nhiều trong các game 3D.

## Chương 5

# Các vấn đề nâng cao về đồ họa 3D trong WebGL

 Nội dung của chương 5 trình bày các vấn đề nâng cao về đồ họa 3D trong WebGL

### 5.1. Hiệu ứng camera

Hiệu ứng camera là việc điều khiển sự di chuyển của camera để tạo ra cho người chơi cảm giác thoải mái trong quan sát hoặc điều khiển camera theo những quỹ đạo di chuyển để tạo cho người chơi những góc nhìn ấn tượng. Nếu hiệu ứng camera được sử dụng một cách hợp lý và khéo léo sẽ tạo cho người chơi cảm giác ấn tượng, thích thú bội phần. Ở phần đầu của mục này sẽ trình bày những vấn đề cơ bản về camera sau đó là các chế độ camera được hỗ trợ trong game.

#### 5.1.1. Cơ bản về camera

##### Vấn đề

Trong đồ họa 3D việc thấy được những gì trên màn hình không phụ thuộc vào việc ta đặt các đối tượng trong thế giới 3D như thế nào mà phụ thuộc vào việc ta đang đặt camera như thế nào. Phần này trình bày những thành phần cơ bản của camera.

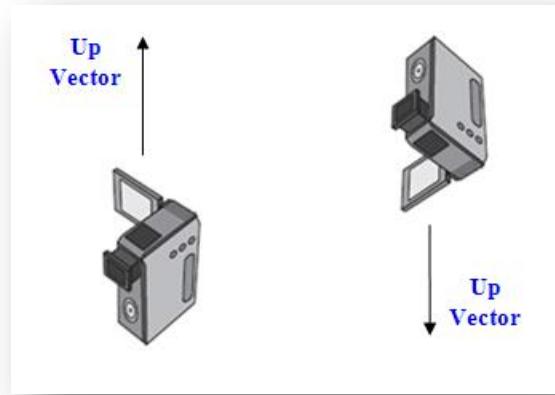
##### Giải pháp

Để thiết lập một camera ta cần 2 ma trận là View và Projection. Ma trận view được khởi tạo từ những thông tin sau:

- **cameraPosition :** Vị trí camera
- **cameraTarget :** Điểm nhìn của camera
- **upVector :** Hướng lên của camera

Dưới đây là một ví dụ về việc tao ma trận view của camera.

```
mat4.lookAt(cameraPosition, cameraTarget, upVector, viewMatrix);
```



Hình 5-1 Minh họa up vector trong Camera

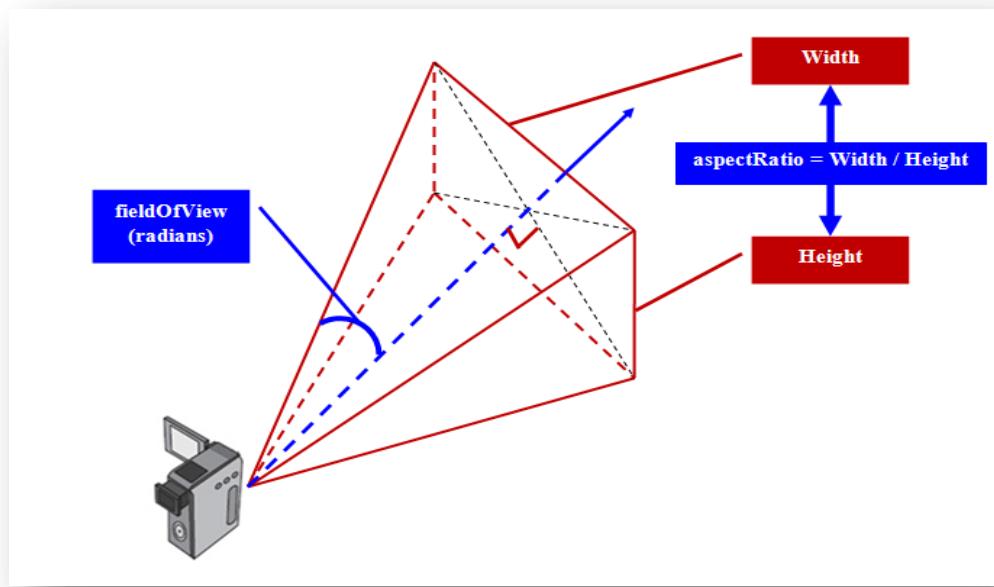
(Nguồn: [3] )

Ma trận projection bao gồm những thông tin sau:

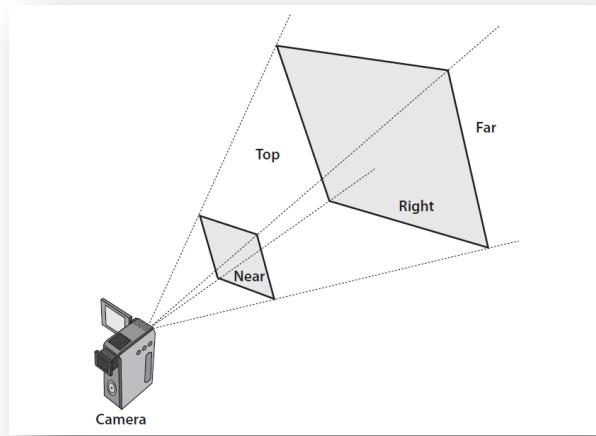
- **fieldOfView**: Độ rộng góc nhìn của camera. Góc mở từ hướng nhìn.
- **aspectRatio**: Tỉ lệ chiều rộng trên chiều cao của màn hình hay mặt phẳng cắt cần hiển thị.
- **nearPlaneDistance**: Khoảng cách đến mặt phẳng gần camera nhất của khoảng có thể nhìn thấy được
- **farPlaneDistance**: Khoảng cách đến mặt phẳng xa camera nhất của khoảng có thể nhìn thấy được

Dưới đây là một ví dụ thiết lập ma trận projection

```
mat4.perspective(fieldOfView, aspectRatio, nearPlane, farPlane, projectionMatrix);
```



**Hình 5-2 Minh họa fieldOfView và aspectRatio**



**Hình 5-3 Minh họa nearPlaneDistance và farPlaneDistance**

(*Nguồn: [3]* )

Projection Matrix định nghĩa một vùng trong không gian 3D mà camera có thể thấy được và chỉ những thứ nằm trong vùng thấy được này mới được vẽ lên màn hình với điều kiện nó không bị che khuất bởi một đối tượng khác cũng trong vùng thấy được này. Những đối tượng nằm ngoài vùng này cho dù nằm trong hướng nhìn của camera cũng không được vẽ lên màn hình.

Chú ý: Nếu để khoảng cách giữa điểm cực cận và điểm cực viễn của camera quá lớn, ta sẽ gặp vấn đề về hiệu suất game vì phải vẽ quá nhiều đối tượng trong không gian quá lớn này mặc dù không hẳn nó đã cần thiết phải vẽ ngay. Tuỳ trường hợp phải làm sao đảm bảo ta chỉ hiển thị đủ những chi tiết cần thiết của thế giới để người chơi cảm thấy hình ảnh hiển thị tự nhiên và loại bỏ các thứ không cần thiết để tránh những vấn đề về hiệu suất của ứng dụng.

### 5.1.2. Chế độ camera theo sau đối tượng

#### Vấn đề

Gặp rất nhiều trong các game action, nhập vai 3D đó là chế độ camera theo sau nhân vật. Đây là chế độ camera với những yêu cầu sau:

- Luôn đi theo theo dõi tượng
- Cho phép nhìn xung quanh đối tượng với tâm quay là đối tượng.
- Cho phép đến gần hoặc ra xa đối tượng

Chúng em đã cài đặt và đóng gói một class **FollowingCamera** hỗ trợ một camera với các chức năng được yêu cầu trên.

#### Giải pháp

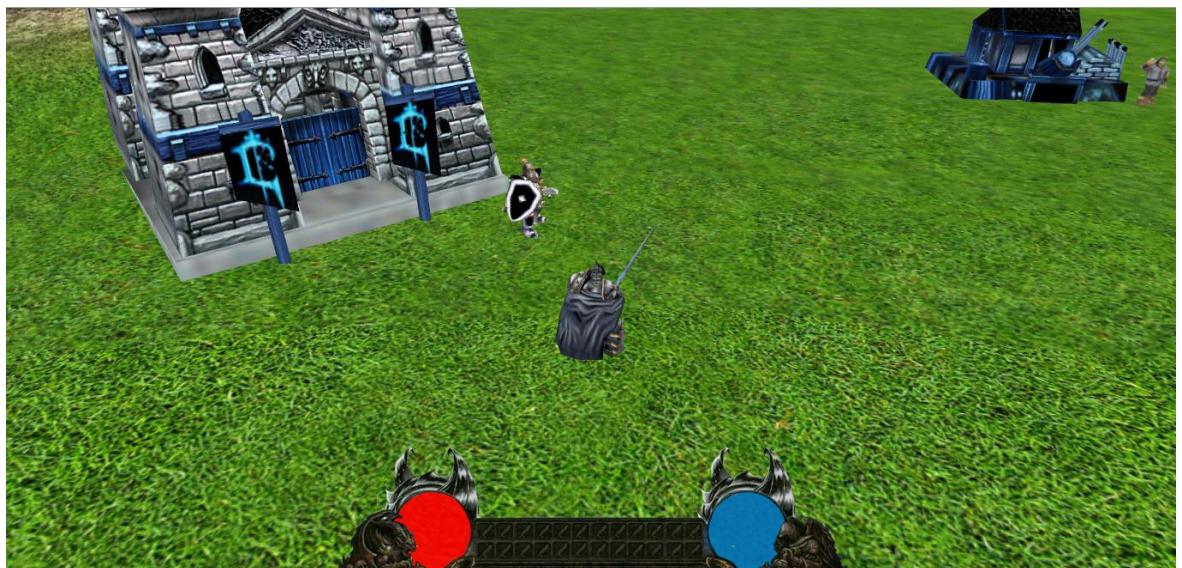
Ban đầu vị trí **FollowingCamera** và vị trí đối tượng cách nhau một khoảng nhất định. Camera target luôn đặt ở vị trí đối tượng. Khi đối tượng di chuyển thì camera cũng di chuyển đúng bằng độ di chuyển của đối tượng.

Hiệu ứng nhìn xung quanh đối tượng được thực hiện bằng một phép quay camera với tâm quay là vị trí đối tượng. Dưới đây là đoạn code minh họa việc quay camera xung quanh đối tượng.

```
var matrix = mat4.create();
mat4.identity(matrix);
mat4.translate(matrix, [this.target.x, this.target.y, this.target.z]);
mat4.rotate(matrix, Util.degToRad(this.rotate), [0, 1, 0]);
mat4.translate(matrix, [-this.target.x, -this.target.y, -this.target.z]);
mat4.multiplyVec3(matrix, [this.position.x, this.position.y, this.position.z]);
```

Lưu ý: **this.target** là vị trí của đối tượng và các ma trận được tính theo thứ tự đảo ngược, tức là trước tiên chúng ta translate theo vector **[-this.target.x, -this.target.y, -this.target.z]**.

`this.target.z]` sau đó thực hiện phép xoay quanh trục y và cuối cùng là translate theo vector `[this.target.x, this.target.y, this.target.z]`



**Hình 5-4 Camera theo sau đối tượng**

## 5.2. Tạo dựng địa hình

Địa hình là một thành phần không thể thiếu trong không gian 3D. Muốn xây dựng bất cứ một Game nào thì thứ đầu tiên cần phải có đó chính là địa hình. Từ địa hình đã được xây dựng, các đối tượng khác mới được đặt lên bề mặt của địa hình này.

Việc xây dựng một địa hình trông ấn tượng và thực tế trong không gian 3D không đơn giản. Cần qua nhiều bước và vận dụng nhiều kỹ thuật để có thể xây dựng được một địa hình trông gần giống với thực tế. Phần này sẽ trình bày về từng bước để có thể xây dựng một địa hình trong game

### 5.2.1. Tạo địa hình từ một ảnh

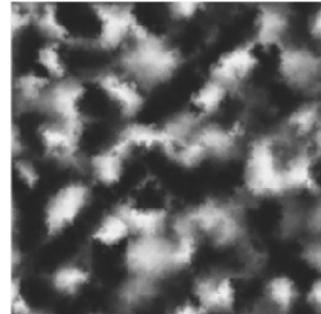
#### 🔒 Ván đè

Bước đầu tiên để xây dựng một địa hình đó là xây dựng một mảng các đỉnh tạo nên địa hình và một mảng index quy định thứ tự các đỉnh để vẽ các primitives tạo thành bề mặt của địa hình từ mảng các đỉnh.



## Giải pháp

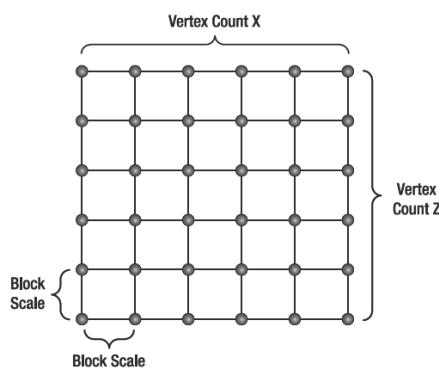
Có nhiều cách để có được 2 thành phần này nhưng trong ứng dụng này chúng em chọn cách tạo dựng địa hình bằng dữ liệu độ cao được lấy từ một ảnh trắng đen.



**Hình 5-5** Ảnh height map

(Nguồn: [3] )

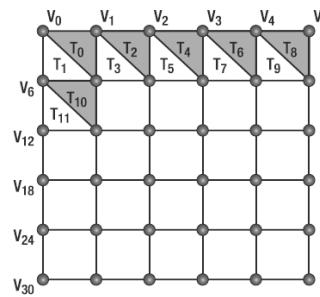
Ảnh trắng đen được sử dụng để dựng địa hình được gọi là *height map*. Chiều rộng của ảnh tương ứng với số lượng đỉnh theo trục x trên địa hình (Vertex Count X), chiều cao của ảnh tương ứng với số lượng đỉnh theo trục z (Vertex Count Z). Độ cao của một đỉnh chính là giá trị màu tại pixel tương ứng trên ảnh. Ví dụ nếu pixel tại dòng 10 cột 10 trên ảnh height map có màu trắng (giá trị là 255) tức là đỉnh tương ứng tại dòng 10 và cột 10 có độ cao là 255. Tương tự nếu pixel có màu đen (giá trị là 0) tức là đỉnh tương ứng trên địa hình có độ cao là 0.



**Hình 5-6** Lưới địa hình

(Nguồn: [3] )

Chúng ta còn sử dụng thêm một biến để lưu khoảng cách giữa hai đỉnh liên tiếp nhau được gọi là *Block Scale*. Như vậy chiều rộng thực tế của địa hình là bằng  $\text{Block Scale} \times (\text{Vertex Count X} - 1)$  tương tự chiều sâu thực tế của địa hình là bằng  $\text{Block Scale} \times (\text{Vertex Count Z} - 1)$ .



**Hình 5-7 Các khối của địa hình**

(Nguồn: [3] )

Mỗi khối trên địa hình liên quan đến 4 đỉnh và được vẽ bằng hai tam giác. Khi vẽ hết được các tam giác thì ta đã dựng được một địa hình. Số lượng tam giác sẽ được tính như sau :

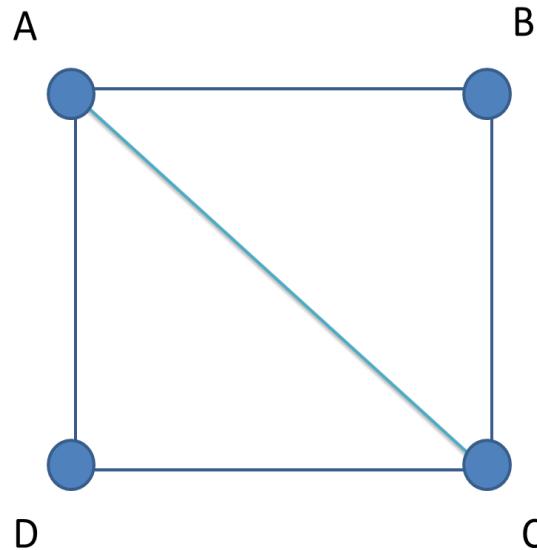
```
this.numTriangles = (this.vertexCountX - 1) * (this.vertexCountZ - 1) * 2;
```

Các đỉnh trên địa hình sẽ được vẽ theo chỉ số. Số lượng chỉ số bằng số lượng tam giác nhân với 3. Dưới đây là đoạn code thực hiện khởi tạo mảng chỉ số (index) .

```
Terrain.prototype.GenerateTerrainIndices = function () {
    var numIndices = this.numTriangles * 3;
    var indices = new Array(numIndices);
    var indicesCount = 0;
    for (var i = 0; i < (this.vertexCountZ - 1); i++) {
        for (var j = 0; j < (this.vertexCountX - 1); j++) {
            var index = j + i * this.vertexCountZ;
            indices[indicesCount++] = index;
            indices[indicesCount++] = index + 1;
            indices[indicesCount++] = index + this.vertexCountX + 1;

            indices[indicesCount++] = index + this.vertexCountX + 1;
            indices[indicesCount++] = index + this.vertexCountX;
            indices[indicesCount++] = index;
        }
    }
    return indices;
}
```

Biến index cho ta biết khối nào đang được xét. Ví dụ xét khối như trong hình Hình 5-8 thứ tự vẽ các đỉnh sẽ là A B C C D A



**Hình 5-8 Thứ tự vẽ các đỉnh trong một khối**

Sau khi đã có mảng chỉ số các đỉnh , việc tiếp theo cần làm là tạo mảng các đỉnh của địa hình. Với địa hình như trong Hình 5-7 thì ta sẽ có một mảng gồm 36 đỉnh. Dưới đây là đoạn code tạo mảng các đỉnh của địa hình.

```
Terrain.prototype.GenerateTerrainVertices = function () {
    var halfTerrainWidth = (this.vertexCountX - 1) * this.blockScale * 0.5;
    var halfTerrainDepth = (this.vertexCountZ - 1) * this.blockScale * 0.5;
    // Texture coordinates
    var tu = 0;
    var tv = 0;
    var tuDerivative = 1.0 / (this.vertexCountX - 1);
    var tvDerivative = 1.0 / (this.vertexCountZ - 1);
    var vertexCount = 0;
    var vertices = new Array(this.vertexCountX * this.vertexCountZ);
    // Set vertices position and texture coordinate
    for(var i=halfTerrainDepth*(-1);i<=halfTerrainDepth;i+=this.blockScale)
    {
        tu = 0;
        for(var j=halfTerrainWidth*(-1);j<=halfTerrainWidth;j+=this.blockScale)
        {
            vertices[vertexCount] = {};
            vertices[vertexCount].Position=[j, this.heightMap[vertexCount], i];
            vertices[vertexCount].TextureCoord=[tu,tv];
            tu += tuDerivative;
            vertexCount++;
        }
    }
}
```

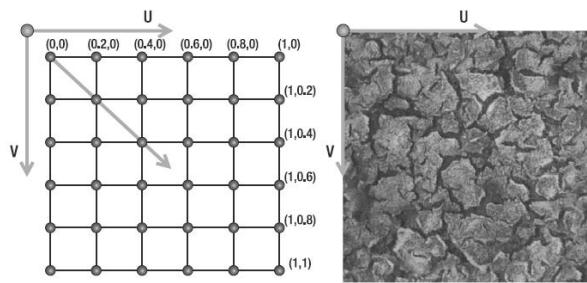
```

    tv += tvDerivative;
}
return vertices;
}

```

Địa hình được tạo sẽ nằm đối xứng qua gốc tọa độ  $O(0,0,0)$  và đó là lý do ta cần tính  $halfTerrainWidth$  (một nửa chiều rộng),  $halfTerrainDepth$  (một nửa chiều sâu) và duyệt danh sách các đỉnh từ  $-halfTerrainWidth$  đến  $+halfTerrainWidth$  và từ  $-halfTerrainDepth$  đến  $+halfTerrainDepth$ .

Thông tin của mỗi đỉnh bao gồm vị trí và tọa độ lấy màu texture lợp địa hình. *this.heightMap* là mảng lưu giá trị màu của tất cả các pixel trên ảnh height map. Hình 5-9 minh họa cách tính tọa độ texture lợp địa hình cho các đỉnh.



**Hình 5-9 Tọa độ texture lợp địa hình**

(Nguồn: [3] )

### 5.2.2. Lợp texture trên địa hình

#### **Vấn đề**

Trong thao tác lợp texture trên địa hình sẽ có 2 vấn đề sau này sinh:

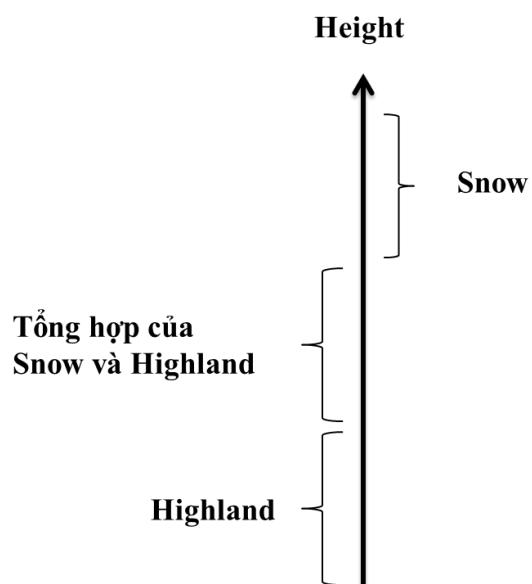
**Vấn đề 1:** Một địa hình không chỉ được lát với một loại texture duy nhất mà thực tế tùy vào độ cao mà sẽ có một loại texture khác nhau ví dụ như trên đỉnh núi cao sẽ có tuyết (texture tuyết), đồng bằng sẽ là cỏ (texture cỏ), ...

**Vấn đề 2:** Với nhiều loại texture khác nhau như vậy cho từng vùng độ cao, sẽ có vấn đề đặt ra là ở những chỗ chuyển tiếp giữa hai loại địa hình sẽ thay đổi như thế nào? Nếu đột ngột chuyển đổi sẽ cho một cảnh hoàn toàn không giống thực tế. Để giống thực tế ở những vùng chuyển tiếp sẽ cần blend cả hai loại texture địa hình với tỷ lệ tương ứng với độ cao hiện tại của vùng chuyển tiếp.



## Giải pháp

Vấn đề 1 và 2 sẽ được giải quyết như sau: Sẽ có nhiều texture để lợp địa hình. Màu tại một điểm trên địa hình được xác định tùy theo độ cao của điểm đó. Cần quy định trước mỗi loại texture được sử dụng cho những khoảng độ cao nào. Tại những điểm nằm giữa hai khoảng độ cao thì màu được xác định là tổng hợp của hai màu texture theo phép nội suy tuyến tính.



**Hình 5-10 Khoảng độ cao của địa hình**

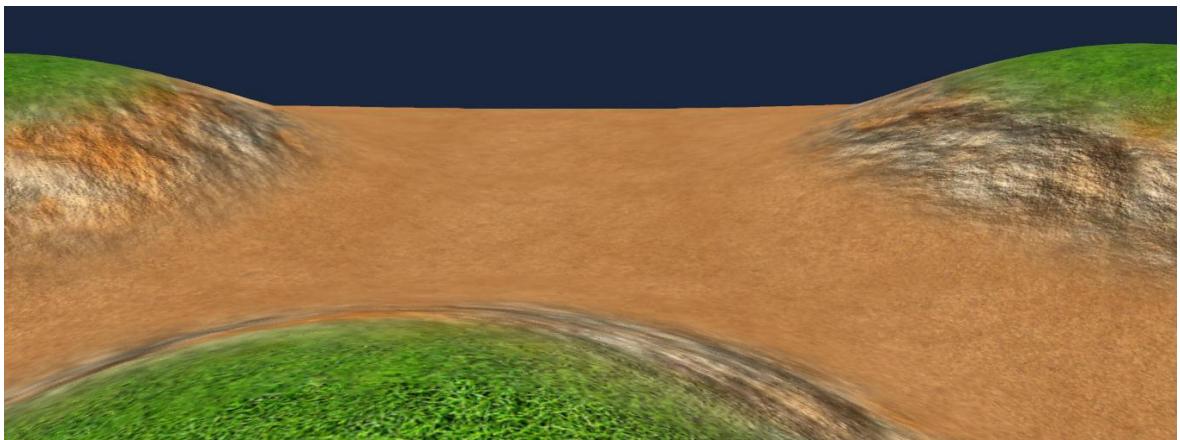
Ví dụ ta quy định các khoảng độ cao như sau:

- Từ 200 đến 250 : Snow
- Từ 100 đến 150 : Highland

Ta xác định màu texture tại độ cao 180 như sau:

$$Color = \left(1 - \frac{180 - 150}{200 - 150}\right) \times Color_{Highland} + \left(\frac{180 - 150}{200 - 150}\right) \times Color_{Snow}$$

Tới đây vấn đề 1 và 2 của việc dán texture cho địa hình đã được giải quyết. Kết quả là tùy độ cao sẽ có một loại địa hình và khoảng tiếp giáp chuyển đổi giữa 2 loại địa hình được chuyển đổi một cách tự nhiên.



**Hình 5-11 Địa hình được lợp bởi nhiều texture**

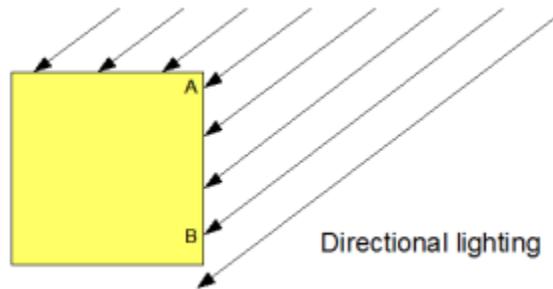
### **5.2.3. Chiếu sáng địa hình**

#### **Ván đề**

Khi ánh sáng mặt trời chiếu lên địa hình từ một hướng, những bề mặt được chiếu sáng nên được làm cho sáng hơn và những bề mặt bị che khuất nên tối hơn. Phần này sẽ đề cập đến kỹ thuật chiếu sáng với ánh sáng song song (từ một nguồn sáng ở rất xa như mặt trời)

#### **Giải pháp**

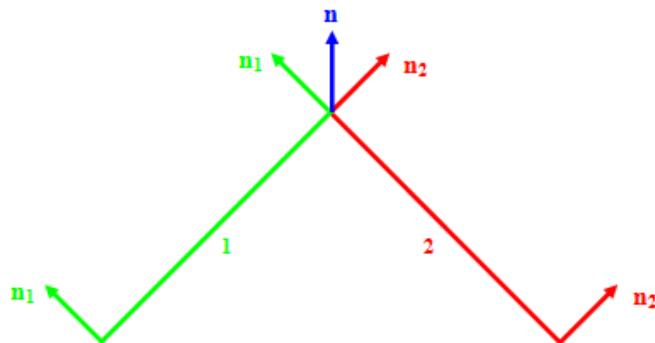
Vậy với một bề mặt làm sao để xác định được bề mặt này sẽ được chiếu sáng với cường độ như thế nào? Một bề mặt có một vector pháp tuyến vuông góc với bề mặt. Dựa vào góc của vector pháp tuyến bề mặt và phương ánh sáng chiếu đến mà xác định cường độ ánh sáng chiếu lên bề mặt. Nếu góc này bằng 0 nghĩa là bề mặt đang vuông góc với phương ánh sáng chiếu tới hay nói cách khác ánh sáng đang chiếu thẳng lên bề mặt, lúc này cường độ ánh sáng chiếu lên bề mặt này là cực đại. Khi góc chiếu càng gần 90 thì ánh sáng chiếu lên bề mặt càng ít, khi pháp tuyến vuông góc với hướng ánh sáng chiếu tới lúc này bề mặt nằm cùng phương với ánh sáng chiếu tới và dĩ nhiên không được chiếu sáng.



**Hình 5-12 Nguồn sáng song song**

Để thực hiện việc chiếu sáng, mỗi đỉnh trong bộ ba đỉnh tạo nên một primitive phải có thông tin về vector pháp tuyến và dựa vào 3 pháp tuyến này khi 3 đỉnh cùng thông tin pháp tuyến này được truyền từ Vertex Shader cho Fragment Shader, GPU sẽ thực hiện việc nội suy ra pháp tuyến của từng pixel trên bề mặt của primitive này và tính toán màu được chiếu sáng tương ứng.

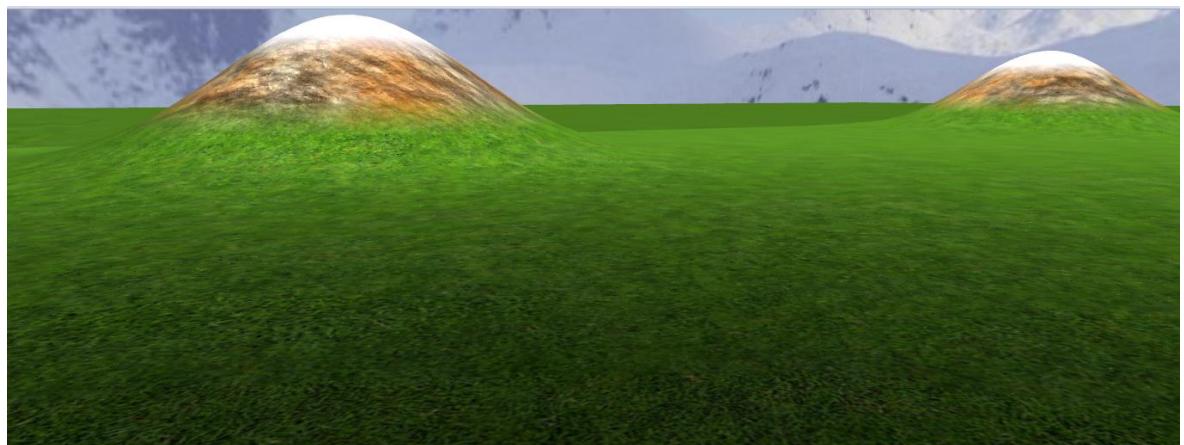
Chú ý nếu một đỉnh nằm trên tiếp giáp giữa nhiều mặt thì pháp tuyến tại đỉnh này sẽ là vector tổng của tất cả pháp tuyến các mặt chứa đỉnh này. Pháp tuyến tổng sẽ được chuẩn hoá về độ dài bằng 1 trước khi truyền vào cho Vertex Shader xử lý.



**Hình 5-13 Vector pháp tuyến tổng**

Phần vừa được bàn ở trên là lý thuyết của kỹ thuật chiếu sáng. Giờ sẽ được áp dụng vào việc chiếu sáng địa hình. Với mỗi đỉnh trong mảng đỉnh tạo nên địa hình phải có thông tin của pháp tuyến và nếu một đỉnh thuộc nhiều mặt có pháp tuyến khác nhau ta cần tính vector tổng của tất cả pháp tuyến của các mặt chứa đỉnh và chuẩn hoá (normalize) pháp tuyến này.

Chúng ta cần truyền thêm các tham số liên quan đến chiếu sáng bao gồm hướng chiếu sáng và độ lớn của ánh sáng xuống để GPU xử lý. Các tham số này nên được truyền như các biến uniform vì được dùng chung cho các vertex khác nhau.



**Hình 5-14 Chiếu sáng địa hình**

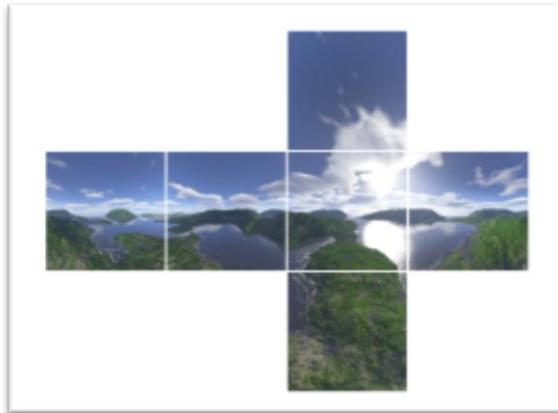
#### **5.2.4. Kỹ thuật Skybox**

##### **Vấn đề**

Trong các game 3D, chúng ta sẽ thấy có những khung cảnh bao quanh như là cảnh bầu trời, núi, nhà. Chính những quang cảnh này làm cho không gian 3D trở nên thật hơn và sống động hơn.

##### **Giải pháp**

Skybox là một kỹ thuật được sử dụng để tạo khung cảnh bao quanh. Ưu điểm của skybox là đơn giản dễ xây dựng và chi phí để vẽ hình thấp. Skybox là một khối hình hộp trong không gian 3D được tạo bởi nhiều texture. Một skybox thường bao gồm 5 đến 6 texture tùy thuộc vào việc nó có chứa mặt đất hay không.



Hình 5-15 Các texture trong skybox

### 5.3. Kỹ thuật Blending



Vấn đề :

Kỹ thuật blending cho phép tạo hiệu ứng trong suốt đẹp mắt trong game 3D. Phần này trình bày cách thực hiện blending trong WebGL.

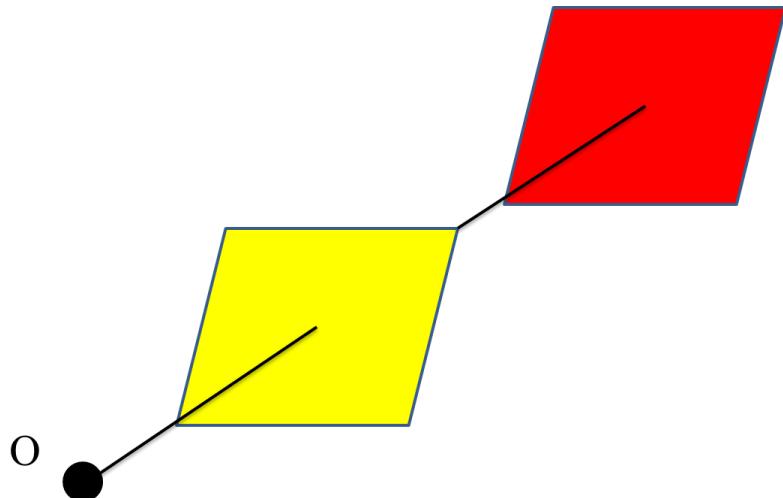


Giải pháp

Để sử dụng blending trong webGL ta cần xác định hàm kết hợp màu (blend) dưới đây là một số ví dụ về hàm blend.

```
gl.blendFunc(gl.SRC_ALPHA, gl.ONE);
gl.blendFunc(gl.SRC_ALPHA, gl.ONE_MINUS_SRC_ALPHA);
gl.blendFunc(gl.CONSTANT_ALPHA, gl.ONE_MINUS_SRC_ALPHA );
```

Một pixel trên màn hình có thể được xác định màu nhiều lần. Ví dụ chúng ta có một hình vuông màu vàng có tâm là (0,0,-5) và một hình vuông thứ hai màu đỏ cùng kích thước có tâm là (0,0,-10), cả hai hình vuông đều vuông góc với trục Ox và vị trí camera nằm tại (0,0,0). Như vậy khi vẽ hình chắc chắn sẽ có vùng giao nhau giữa hai hình và những pixel tại vùng giao nhau sẽ lấy màu của hình vuông gần hơn tức là màu vàng.



**Hình 5-16 Minh họa vấn đề trong kỹ thuật Blending**

Trong WebGL chúng ta có các khái niệm về màu nguồn (source) và màu đích (destination). Màu đích chính là màu đã có sẵn (trong ví dụ Hình 5-16 là màu đỏ), màu nguồn là được vẽ thêm vào (trong ví dụ Hình 5-16 là màu vàng). Mục tiêu của hàm blend màu là xác định mối quan hệ giữa màu đích và màu nguồn để tính được màu kết quả.

Với hàm blend như sau :

```
gl.blendFunc(gl.SRC_ALPHA, gl.ONE_MINUS_SRC_ALPHA);
```

Màu kết quả sẽ được tính như sau :

$$Color_{result} = Color_{source} \times Alpha_{source} + Color_{destination} \times (1 - Alpha_{source}) \quad (1)$$

Với hàm blend như sau :

```
gl.blendFunc(gl.SRC_ALPHA, gl.ONE);
```

Màu kết quả tương ứng sẽ là :

$$Color_{result} = Color_{source} \times Alpha_{source} + Color_{destination} \quad (2)$$

Với hàm blend (1) thì màu kết quả sẽ phục thuộc vào giá trị kênh alpha của màu nguồn và nó sẽ quyết định độ trong suốt. Trong trường hợp hàm blend (2) thì độ trong suốt sẽ nhiều hơn vì giá trị alpha luôn nhỏ hơn hay bằng một.

Như vậy kỹ thuật blending phụ thuộc giá trị alpha của màu nguồn và cách xác định giá trị alpha này như thế nào ? Dưới đây là một đoạn code GLSL của Fragment shader

```
precision mediump float;
varying vec2 vTextureCoord;
uniform sampler2D uSampler;
void main(void) {
    vec4 gl_FragColor = texture2D(uSampler, vec2(vTextureCoord.s,
vTextureCoord.t));
}
```

Nhắc lại mục đích của Fragment shader đó là xác định màu của vertex. Trong đoạn code trên thì màu cần xác định là màu tại tọa độ (*vTextureCoord.s*, *vTextureCoord.t*) của texture *uSampler*. Như vậy nếu texture được sử dụng có hỗ trợ kênh alpha (ảnh .png) thì màu của vertex sẽ có kênh alpha. Ngoài ra chúng ta cũng có khả năng thay đổi giá trị alpha của vertex, dưới đây là một ví dụ minh họa việc đó.

```
precision mediump float;
varying vec2 vTextureCoord;
uniform sampler2D uSampler;
void main(void) {
    vec4 textureColor = texture2D(uSampler, vec2(vTextureCoord.s,
vTextureCoord.t));
    if(textureColor.r <= 0.5 && textureColor.g <= 0.5 && textureColor.b <=
0.5)
        gl_FragColor = vec4(textureColor.rgb, 0.0);
    else
        gl_FragColor = vec4(textureColor.rgb, 1.0);
}
```



**Hình 5-17 Kỹ thuật Blending tạo hình ảnh trong suốt**

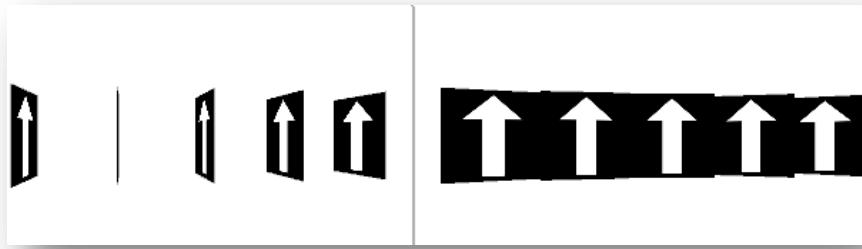
#### **5.4. Kỹ thuật Billboard**

##### **🔒 Vấn đề :**

Billboard là một kỹ thuật sử dụng nhiều trong đồ họa 3D để thực hiện một số hiệu ứng đẹp mắt. Phần này sẽ trình bày chi tiết về kỹ thuật Billboard được áp dụng trong game 3D.

##### **🔑 Giải pháp :**

Billboarding là kỹ thuật thay vì phải render cả một mô hình 3D tại một vị trí trong không gian thì chỉ cần vẽ một ảnh texture của đối tượng cần hiển thị tại vị trí đó sao cho texture này luôn đối mặt với camera khi camera di chuyển. Điều này sẽ tiết kiệm chi phí rất nhiều trong việc xử lý không gian ngoại cảnh và giúp ứng dụng game tăng tốc đáng kể.



**Hình 5-18 Kỹ thuật billboard**

*Ảnh trái: Vẽ texture 2D trong không gian 3D bình thường*

*Ảnh phải: Vẽ texture 2D trong không gian 3D áp dụng billboarding*

Trong kỹ thuật Billboard thông thường việc tính toán được thực hiện trong Vertex Shader lập trình bằng GLSL. Tất cả các đỉnh tạo nên hình chữ nhật render texture được truyền vào Vertex Shader với **cùng một vị trí giống nhau**, việc tính toán ra các vị trí đúng của các đỉnh sẽ được thực hiện trong Vertex Shader dựa trên hướng nhìn từ camera và tọa độ texture của các đỉnh. Dưới đây là định nghĩa của lớp BillboardModel được sử dụng để thực hiện kỹ thuật billboard.

```
function BillboardModel() {
}
BillboardModel.inheritsFrom(GameModel);
BillboardModel.prototype.load = function(textureID, effectID) {
    var vertexTextureCoords =[  
        0.0, 0.0,  
        1.0, 0.0,  
        1.0, 1.0,  
        0.0, 1.0  
    ];  
    var vertexPositions = [  
        0.0, 0.0, 0.0,  
        0.0, 0.0, 0.0,  
        0.0, 0.0, 0.0,  
        0.0, 0.0, 0.0  
    ];  
    var indices = [0, 1, 2, 0, 2, 3];
    //...
}
```

Billboard được khởi tạo bao gồm 4 vertex có tọa độ là  $(0,0,0)$  với tọa độ texture như trong đoạn code trên. Các vertex sẽ được vẽ theo chỉ số để tạo được một hình chữ nhật.

Trong phần code GLSL ta cần thêm thông tin về vị trí camera, vector up và tỷ lệ scale của kích thước billboard. Dưới đây là đoạn code GLSL của vertex shader.

```

attribute vec3 aVertexPosition;
attribute vec2 aTextureCoord;
uniform mat4 uVMMatrix;
uniform mat4 uPMatrix;
uniform mat4 uWMatrix;
varying vec2 vTextureCoord;

uniform vec3 cameraPosition;
uniform vec3 upVector;
uniform float scale;
void main(void) {
    vec3 center = uWMatrix*vec4(aVertexPosition, 1.0).xyz;

    vec3 eyeVector = center - cameraPosition;
    vec3 sideVector = cross(eyeVector,upVector);
    sideVector = normalize(sideVector);

    vec3 finalPosition = center;
    finalPosition += (aTextureCoord.x-0.5)*scale*sideVector;
    finalPosition += (aTextureCoord.y-0.5)*scale*upVector;

    vec4 finalPosition4 = vec4(finalPosition, 1.0);
    gl_Position = uPMatrix*uVMMatrix*finalPosition4;
    vTextureCoord = aTextureCoord;
}

```

Trước tiên, ta cần tính vị trí của tâm billboard bằng cách nhân vị trí gốc với ma trận world. Tiếp theo cần xác định vector từ tâm billboard đến vị trí camera gọi là *eyeVector*. Sau đó ta tính *sideVector* là tích có hướng của *eyeVector* và *upVector*. Tiếp theo ta đã có thể xác định được vị trí sau cùng của đỉnh dựa vào tọa độ texture *scale*, *upVector* và *sideVector*.



## Hình 5-19 Kỹ thuật Billboard

### 5.5. Kỹ thuật Particle



#### Vấn đề :

Đối với một ứng dụng 3D, đặc biệt là game 3D, điều cần quan tâm nhất là thiết kế game sao cho đẹp mắt, dễ sử dụng, khiến người chơi thấy thích thú khi chơi game. Để làm được điều này, ngoài khâu thiết kế các đối tượng nhân vật, nhà cửa trong game, cần chú ý đến các hiệu ứng như : khói lửa, hòa quang của nhân vật, sương mù, hiệu ứng nổ, chưởng, phép thuật.... Để đạt được những hiệu ứng này, chúng ta cần áp dụng kỹ thuật Partilce.



#### Giải pháp :

##### ❖ Cơ bản về kỹ thuật Particle:

- Lúc bắt đầu hiệu ứng, sẽ có rất nhiều particle tại vị trí tâm, do đó vị trí tâm sẽ sáng nhất vì màu của các particle cộng hưởng với nhau.
- Mỗi particle sẽ di chuyển theo các hướng ngẫu nhiên với vận tốc, độ sáng, kích thước, độ xoay khác nhau sau một khoảng thời gian nhất định.

##### ❖ Sử dụng particle:

- Khi sử dụng particle ta cần sử dụng một số biến bao gồm số lượng particle, thời điểm sinh ra particle, thời gian tồn tại của particle, thời điểm hiện tại.

```
function Particle(){  
}  
Particle.prototype.load = function(){  
    this.nParticles = 20;    //so luong particle  
    this.birthTime = 0;      //thoi diem sinh ra particle  
    this.maxAge = 1000;     //thoi gian ton tai  
    this.currentTime = 0;    //thoi diem hien tai  
    //...  
}
```

- Trong ví dụ trên các biến *birthTime*, *maxAge* và *currentTime* sẽ được dùng để xác định vị trí và giá trị alpha của từng particle. Mỗi particle về cơ bản là một hình chữ nhật sử dụng kỹ thuật billboard, tức là luôn quay theo hướng nhìn của camera. Particle thường có kích thước nhỏ. Trong lớp Particle của nhóm em cài đặt thì các particle được lưu thành một mảng các đỉnh, khi vẽ tất cả các đỉnh thì ta đạt được hiệu ứng particle. Dưới đây là đoạn code GLSL minh họa vertex shader của hiệu ứng particle.

```

void main(void) {
    vec3 startingPosition = uWMatrix*vec4(aVertexPosition, 1.0).xyz;

    float age = currentTime - birthTime;
    float relAge = age/maxAge;

    float totalDisplacement = relAge*40.0;
    vec3 billboardCenter = startingPosition+totalDisplacement*aMoveDirection;

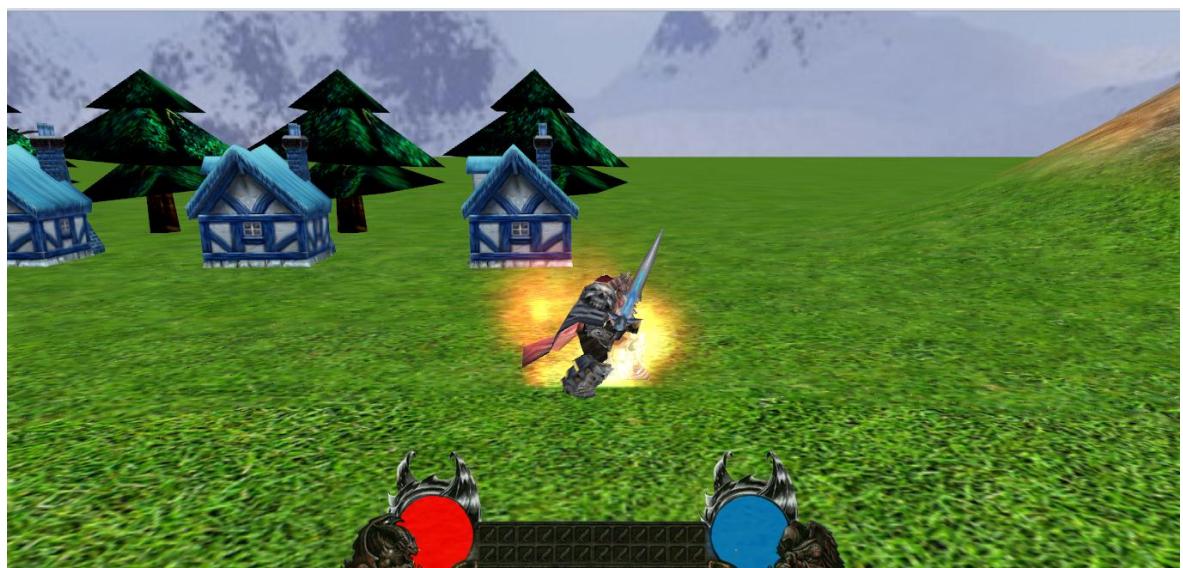
    //...
}

```

Đầu tiên chúng ta xác định vị trí tâm hiện tại *startingPosition*. Sau đó chúng ta tính độ tuổi tương đối của particle so với thời gian sống của nó. Tiếp theo ta xác định vị trí mới gọi là *billboardCenter* bằng cách cộng *startingPosition* với tích của *totalDisplacement* và *aMoveDirection* (*aMoveDirection* là hướng di chuyển random của mỗi particle). Tiếp theo ta thực hiện tương tự hiệu ứng billboard với *billboardCenter* đã có.

### ❖ Kết luận

Từ kỹ thuật Particle System, chúng ta có thể tạo ra được các hiệu ứng Particle tùy ý như : khói lửa, nổ, chưởng. Các khâu xử lý đều nằm ở tầng dưới, tức là VertexShader, nên đỡ tốn chi phí hơn nếu chúng ta xử lý ở tầng cao hơn.



**Hình 5-20 Kỹ thuật Particle tạo hiệu ứng lửa quanh nhân vật**

## 5.6. Kỹ thuật Ray Picking



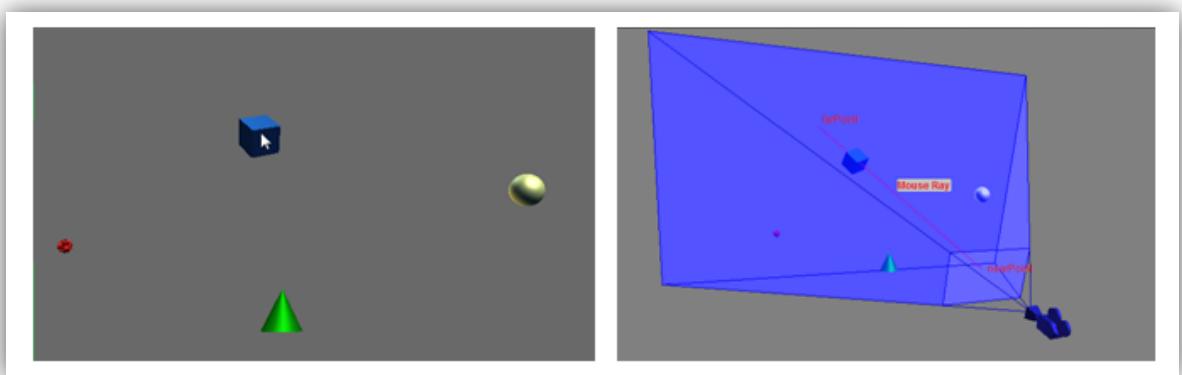
### Vấn đề :

Đối với một game 3D, mạng xã hội 3D, hay đối với một ứng dụng trên môi trường 3D nói chung, nhu cầu về tương tác lựa chọn một vật thể bất kỳ trong môi trường 3D là việc rất cần thiết và rất thường xuyên. VD : chọn quyển sách trong một thư viện sách 3D, chọn một quái vật bất kỳ để thực hiện thao tác đánh quái vật đó trong game 3D, ... việc lựa chọn này có thể dùng phím keyboard để thao tác chọn vật thể bằng cách di chuyển lại gần vật thể hoặc va chạm với vật thể, nhưng nó thật sự không hữu dụng đối với một số nhu cầu khác. Nhu cầu bắn súng đến vị trí nào đó hoặc chưởng phép đến mục tiêu nào đó, lúc này keyboard sẽ không đáp ứng được một cách hoàn hảo. Vì thế, việc chọn một vật thể hay xác định một đối tượng trong môi trường 3D bằng click chuột ra đời.



### Giải pháp :

Việc lựa chọn một vật thể hay một đối tượng trong môi trường 3D bằng click chuột được hỗ trợ trên hầu hết các engine 3D. Và có nhiều cách để thực hiện việc này. Nhóm chúng em đã tham khảo nhiều cách và đã lựa chọn cách Ray Picking để lựa chọn vật thể trong môi trường 3D. Nhiều engine đã sử dụng cách này để thực hiện chức năng picking, điển hình là XNA, ThreeJS, SceneJS...



Hình 5-21 Minh họa RayPicking thể hiện ở 2 kiểu Camera

Các bước cài đặt chức năng Ray Picking :

- **Bước 01** : Chuyển trục tọa độ click chuột ra giữa màn hình.
- **Bước 02** : Xác định tọa độ farPoint ứng với vị trí click chuột.

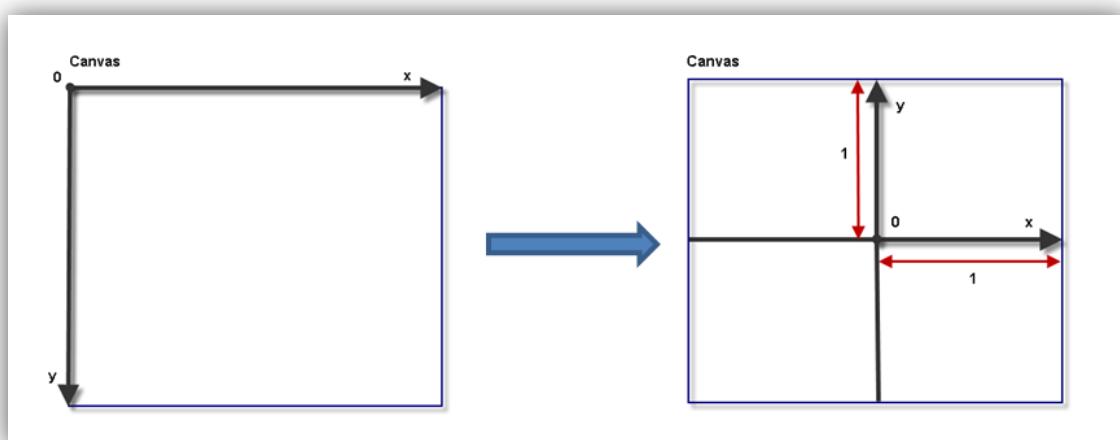
- **Bước 03** : Xác định vật thể nằm trên hướng (startPoint, farPoint).

### (1) Chuyển trực tọa độ click chuột ra giữa màn hình

Trục tọa độ mặc định của thẻ canvas nằm ở góc trên trái của thẻ. Chuẩn hóa gốc tọa độ ra giữa thẻ canvas và kích thước của thẻ có độ dài là 1. Được chuẩn hóa bởi công thức :

```
x = (mouseX / canvas.width) * 2 - 1;
y = 1 - (mouseY / canvas.height) * 2;
```

- **mouseX, mouseY** : tọa độ click chuột lên canvas
- **canvas.width, canvas.height** : kích thước thẻ canvas
- **x, y** : tọa độ click chuột sau khi chuyển gốc tọa độ



**Hình 5-22** Minh họa gốc tọa độ click chuột trước và sau khi chuyển đổi.

### (2) Xác định tọa độ FarPoint ứng với vị trí click chuột

Các engine tích hợp RayPicking đều cài đặt hàm **Unproject** để xác định tọa độ FarPoint từ tọa độ click chuột. Cụ thể như sau :

```
Projector.prototype.pickingRay = function(mouseX, mouseY, camera)
{
    var farPoint = new Vector3( mouseX, mouseY, 1.0 );
    farPoint = this.unprojectVector(farPoint, camera );

    startPoint = new Vector3(camera.position);
    var ray = new Ray(startPoint, farPoint);
    return ray;
}

Projector.prototype.unprojectVector = function(vector, camera)
{
    var screenMatrix = new Matrix4();
    var projectMatrixInverse = new Matrix4();
    projectMatrixInverse.getInverse( camera.pMatrix );
```

```

        screenMatrix.multiply( camera.vMatrix, projectMatrixInverse );
        screenMatrix.multiplyVector3( vector );
        return vector;
    }

```

- ***farPoint*** : tọa độ điểm farPoint sau khi được xử lý qua hàm **unprojectVector** ta được tọa độ farPoint nằm trên mặt phẳng far của projector.
- ***startPoint*** : tọa độ điểm bắt đầu
- ***ray*** : trả về một tia ray có tọa độ điểm bắt đầu (tọa độ camera) và tọa độ hướng của tia là farPoint.

### (3) Xác định vật thể nằm trên hướng (**startPoint, farPoint**)

Để xác định tia Ray vừa tạo ra có chiếu qua vật thể hay không thì có nhiều cách thực hiện.

- Xác định từng tọa độ vị trí theo một khoảng nhất định trong đoạn thẳng [startPoint, farPoint] xem có thuộc boundingbox của vật thể không, nếu có thì tia Ray này chiếu qua vật thể
- Xét khoảng cách từ vị trí tâm của vật thể đến đường thẳng đi qua 2 điểm startPoint, farPoint. Bằng toán học lớp 12, tính khoảng cách từ 1 điểm đến 1 đường thẳng, nếu nhỏ hơn một khoảng cho trước thì ta xác định vật thể này nằm trên tia Ray.

## 5.7. Xử lý va chạm



### Vấn đề :

Vấn đề va chạm là không thể thiếu đối với một ứng dụng 3D, và càng không thể thiếu trong một Game 3D. Va chạm giữa nhân vật với các vật thể khác trong không gian 3D, va chạm giữa các nhân vật với nhau. Để cho Game được mượt và thật hơn khi di chuyển thì chúng ta phải nhận biết được khi nào các đối tượng va chạm với nhau để có thể xử lý theo ý mình muốn.



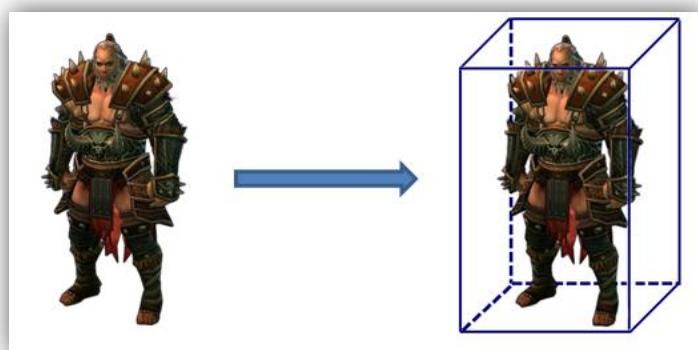
### Giải pháp :

Muốn xử lý va chạm giữa các vật thể trong không gian 3 chiều, trước hết, chúng ta phải xác định được vật thể đó đang đứng ở vị trí nào và chiếm một không gian bao nhiêu. Muốn làm được điều đó, nhóm chúng em đã thiết lập theo các bước sau đây :

- **Bước 01** : Xác định boundingbox của mô hình 3D
- **Bước 02** : Xác định không gian thực mà mô hình chiếm trong không gian 3D
- **Bước 03** : Kiểm tra va chạm với mô hình khác.

## (1) Xác định boundingbox của mô hình 3D

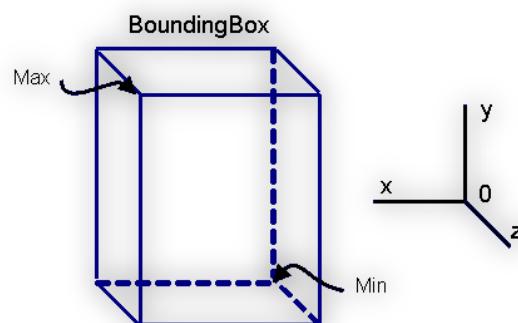
Mỗi một mô hình 3D đều có danh sách các đỉnh vertex để cấu thành. Từ danh sách các đỉnh này, chúng ta có thể tính được tọa độ đỉnh min và max của mô hình đó. Nói một cách tổng quát hơn, chúng ta có thể tạo ra một khung hình hộp bao quanh mô hình 3D.



**Hình 5-23 Minh họa BoundingBox của một mô hình 3D**

Như vậy, cấu trúc của một BoundingBox như sau :

- min : tọa độ điểm nhỏ nhất
- max : tọa độ điểm lớn nhất



**Hình 5-24 Cấu trúc của BoundingBox**

## (2) Xác định boundingbox của mô hình 3D

Để xác định được một mô hình 3D đang chiếm một không gian nào đó trong không gian 3 chiều. Chúng ta có cách tính như sau :

```
var min = new Vector3();
min.x = boundingBox.min.x * scale.x + position.x;
min.y = boundingBox.min.y * scale.y + position.y;
```

```

min.z = boundingBox.min.z * scale.z + position.z;

var max = new Vector3();
max.x = boundingBox.max.x * scale.x + position.x;
max.y = boundingBox.max.y * scale.y + position.y;
max.z = boundingBox.max.z * scale.z + position.z;

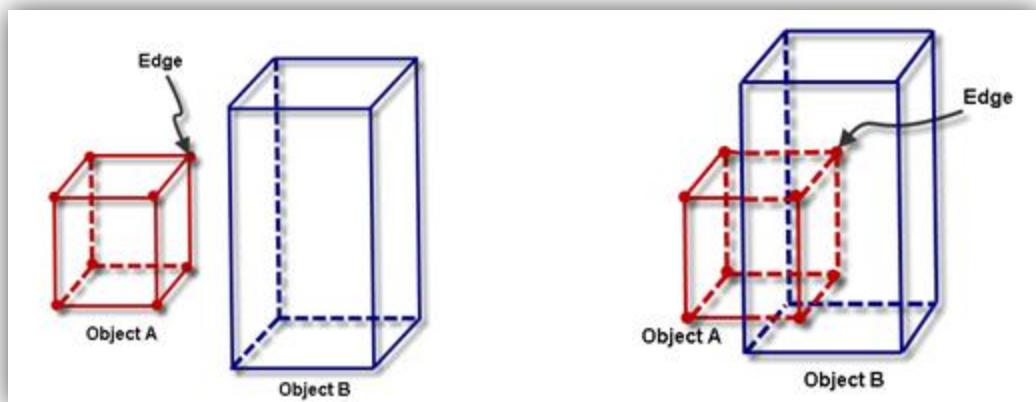
```

- ***boundingBox*** : là BoundingBox của mô hình.
- ***scale*** : là tỉ lệ kích cỡ mô hình
- ***position*** : vị trí hiện tại mà mô hình đang giữ trong không gian 3 chiều.
- ***min, max*** : là tọa độ nhỏ nhất và lớn nhất của mô hình trong không gian 3 chiều.

### (3) Kiểm tra va chạm giữa 2 mô hình 3D

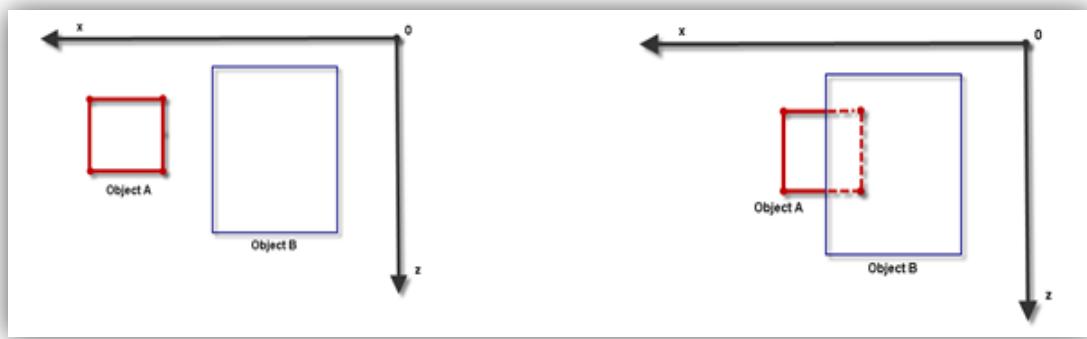
Giờ đây, chúng ta muốn biết được hai mô hình 3D có va chạm với nhau hay không thì chỉ cần xét tọa độ của mô hình này xem có nằm trong boundingbox của mô hình kia hay không.

Biết được 2 đối tượng va chạm với nhau nhưng nó không được thật cho lắm vì chúng ta mới chỉ xét tọa độ của ObjectA va chạm với ObjectB. Muốn cho game trở nên thật hơn, chúng ta xét 8 đỉnh của ObjectA xem có nằm trong vùm không gian của ObjectB hay không. Nếu có thì ObjectA va chạm với ObjectB.



**Hình 5-25 Minh họa 2 Object 3D trước và sau khi va chạm**

Về mặt lý thuyết là vậy, nhưng đối với Game của chúng em, va chạm giữa các mô hình 3D với nhau đều dựa theo 2 chiều X và Z. Va chạm theo chiều Y hầu như là không có. Vì thế, để tối ưu hóa và để cho Game được nhanh hơn. Chúng em chỉ xét va chạm dựa theo 2 chiều X và Z.



**Hình 5-26 Minh họa 2 Object 3D trước và sau khi va chạm theo chiều X, Z**

## 5.8. Hỗ trợ đa người dùng



### Vấn đề :

Nhu cầu chơi game để giải trí là một vấn đề không thể thiếu trong xã hội tân tiến ngày nay. Và nhu cầu các game thủ chơi chung với nhau một màn chơi hay các game thủ thi đấu với nhau cũng là nhu cầu không thể thiếu. Vì thế, đa số các game online/offline hiện nay đều hỗ trợ multiplayer (hỗ trợ đa người dùng)



### Giải pháp :

Để giải quyết vấn đề trên, có thể được hiểu tổng quát như sau : chúng ta sẽ có 2 bộ phận (Client – Server). Client : là phía người dùng. Server là phía quản lý dữ liệu. Khi một Client thay đổi thông tin, Client đó sẽ thông báo đến Server, Server cập nhật dữ liệu của Client đó và truyền dữ liệu mới về cho các Client còn lại. Có nhiều cách để quản lý mã nguồn theo kiểu Client-Server này : ASP.net, Nodejs... Ở đây, nhóm chúng em dùng NodeJS



**Hình 5-27 Chức năng Multiplayer trong Game (Devil)**

Các bước cài đặt cơ chế đa người dùng trên NodeJS :

- **Bước 01** : Cài đặt NodeJS và các gói dữ liệu cần thiết
- **Bước 02** : Thiết lập các hàm để connect giữa Client-Server
- **Bước 03** : Truyền nhận tin giữa Client-Server

### (1) Cài đặt NodeJS và các gói dữ liệu cần thiết

Tải và cài đặt NodeJS v0.6.19 : <http://nodejs.org/dist/v0.6.19/node-v0.6.19.msi>

Sử dụng NPM (Node Package Manager) của Node để cài đặt các gói dữ liệu. Cần cài đặt các gói dữ liệu sau : Express (web framework), Jade (template engine), SocketIO (quản lý kết nối).

Trước hết, chúng ta nên tạo một thư mục mới để chứa các gói dữ liệu. Để cài đặt các gói dữ liệu vào thư mục này, chúng ta vào cmd trả đến thư mục và gõ các lệnh cài đặt sau :

```
npm install express
npm install jade
npm install socket.io
```

Gõ lệnh sau để tạo project mới :

```
express projectname
```

Để thực thi chương trình, chúng ta trả đến thư mục chứa project và gõ :

```
node app.js
```

## (2) Thiết lập các hàm để connect giữa Client-Server :

Thiết lập các hàm cần thiết ở phía Server để lắng nghe connect từ Client

```
var app = express();
Var http = require('http');
var server = http.createServer(app);
var io = require('socket.io').listen(server);
var multiplayer = io
    .of('/multiplayer')
    .on('connection', function (socket)
    {
    });
});
```

Phía Client cần import file socket.io.js để có thể sử dụng các API của gói Socket.io

```
<script type="text/javascript" src="/socket.io/socket.io.js"></script>
```

Client connect đến Server qua thông điệp connect :

```
var socket = io.connect('http://localhost/multiplayer');
socket.on('connect', function ()
{
});
```

## (3) Truyền nhận tin giữa Client-Server :

Giờ đây muốn truyền nhận tin giữa Client-Server là điều rất dễ dàng. Chỉ vài cú pháp sau :

Client emit lên Server :

```
// Client
socket.emit('sendmessage', {id : socketID, message : 'hello'});

// Server
socket.on('sendmessage', function(data)
{
    Console.log(data.message); // show log : 'hello'
});
```

Server truyền thông điệp xuống Client :

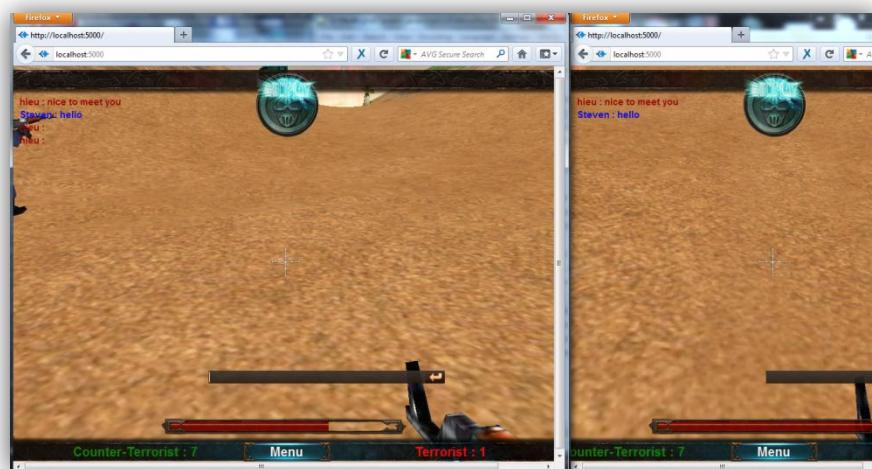
```
// Server
multiplayer.sockets[socketID].emit('BroadcastMessage', data);

// Client
socket.on('BroadcastMessage', function(data)
{
    // receive data from server
});
```

### ❖ Kết luận :

Từ cách truyền nhận tin này, chúng ta có thể thực hiện chức năng **multiplayer** một cách dễ dàng bằng cách truyền nhận thông tin vị trí, góc quay, animation của nhân vật lên Server, sau đó broadcast xuống các Client khác.

Ngoài ra, chúng ta còn có thể thực hiện được chức năng **chat** giữa các người chơi với nhau trong một phòng chơi bằng cách truyền nhận tin input từ người chơi, sau đó truyền nhận thông tin như đã làm ở bước chức năng multiplayer.



Hình 5-28 Minh họa Chat giữa các người chơi với nhau trong Game (GunPlay)

### 5.9. Kết luận

Nội dung của chương này trình bày các vấn đề nâng cao liên quan đến đồ họa 3D trong WebGL. Các kỹ thuật và hiệu ứng được trình bày giúp ta có khả năng tạo ra những game 3D đẹp mắt, hấp dẫn người chơi.

## Chương 6

# Kiến trúc xử lý kịch bản và màn chơi trong game

 Nội dung của chương này trình bày các vấn đề về kiến trúc xử lý kịch bản, quản lý màn chơi, quản lý các thuộc tính và thể hiện lời thoại trong game.

## 6.1. Kiến trúc xử lý kịch bản

### Vấn đề

Trong thể loại game nhập vai một yếu tố cần thiết phải có đó là kịch bản. Nhưng để có thể dễ dàng thay đổi, thêm mới kịch bản trong game mà không cần phải xây dựng lại game thì ứng dụng cần phải có một kiến trúc chương trình thích hợp và một cấu trúc đặc tả cho kịch bản được lưu trữ bên ngoài. Hai yếu tố này kết hợp lại sẽ cho thành quả là ứng dụng game có thể hoàn toàn độc lập với chi tiết kịch bản, việc duy nhất cần làm là thiết kế một kịch bản và tạo lập một file kịch bản theo cấu trúc được quy định dựa trên kịch bản được thiết kế đó, ứng dụng game lúc này sẽ đọc kịch bản từ file và tạo màn chơi với các nhiệm vụ như đã được thiết kế trong kịch bản. Phần này chúng em sẽ trình bày về cấu trúc của file đặc tả cho kịch bản cần tạo dựng và kiến trúc chúng em cài đặt cho việc xử lý kịch bản để giải quyết vấn đề này.

### Giải pháp

Trong chương trình ta cần quản lý một danh sách toàn cục lưu các biến điều kiện và một danh sách khác quản lý toàn bộ các đối tượng trong game. Trong chương trình chúng em đặt tên cho hai danh sách trên lần lượt là **global.Condition** và **global.GameEntity**. **global.GameEntity** cho phép ta thao tác trực tiếp đến những đối tượng được quản lý trong game. Như vậy phần kịch bản của game về cơ bản là phần kiểm tra biến điều kiện và nếu điều kiện thỏa thì thực hiện thao tác lên các đối tượng có thể được lưu bên ngoài file đặc tả. Việc cần làm tiếp là định nghĩa một file đặt tả kịch bản và các thao tác có thể thực hiện khi điều kiện được thỏa.

### 6.1.1. Cấu trúc file kịch bản

Phân đặc tả cho các sự kiện sẽ được lưu trong file xml. Dưới đây là ví dụ về nội dung của một file đặc tả sự kiện:

```
<events>
<event id="teleporttovillage">

    <preconditions>
        <condition id="myhero_go_mapadventure-teleport-mapvillage" type="equal" value="0"/>
        <condition id="myhero_kill_pitlord" type="greaterthan" value="2"/>
    </preconditions>

    <postactions>
        <postaction type="changeMap">
            <parameter destination="mapvillage"/>
        </postaction>
        <postaction type="updateCondition">
            <parameter type="set" targetid="myhero_go_mapadventure-teleport-mapvillage" value="-1"/>
        </postaction>
    </postactions>

    </event>
</events>
```

Trong file đặc tả sẽ bao gồm nhiều event, mỗi event bao gồm những thành phần sau đây:

#### (1) Preconditions

Mỗi node Event sẽ bắt đầu bằng node **preconditions** biểu diễn cho các điều kiện để sự kiện này được kích hoạt. **Preconditions** là một tập hợp các điều kiện so sánh giữa các biến điều kiện *global.Condition* với các giá trị cụ thể. Các phép so sánh được chúng em hỗ trợ bao gồm lớn hơn, nhỏ hơn, hay bằng.

Xét sự kiện *teleporttovillage* trong ví dụ trên, đây là sự kiện khi nhân vật di chuyển sang map khác. **Preconditions** của sự kiện này yêu cầu biến điều kiện *global myhero\_go\_mapadventure-teleport-mapvillage* phải có giá trị là 0 và biến điều kiện *myhero\_kill\_pitlord* phải có giá trị lớn hơn 2 để sự kiện được xảy ra. Biến *myhero\_go\_mapadventure-teleport-mapvillage* cho biết nhân vật có đi đến đúng vị trí yêu cầu để có thẻ *teleport* không, biến *myhero\_kill\_pitlord* cho biết số lượng pitlord đã bị diệt bởi nhân vật.

#### (2) Postactions

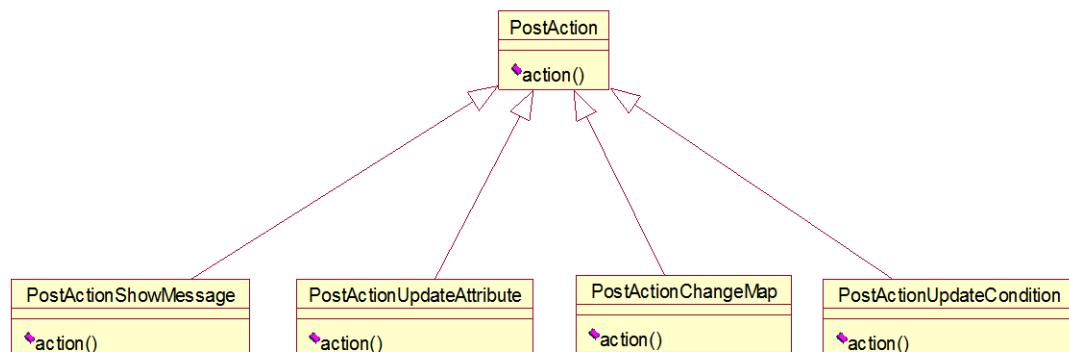
Cuối mỗi node Event bắt buộc phải có một node **postactions**. Node này có nhiệm vụ giúp xác định hành động được thực hiện khi event xảy ra. Có nhiều loại hành động có thể được thực hiện như cập nhật biến điều kiện global, cập nhật thuộc tính của nhân vật, xuất thông báo ra màn hình. Xét sự kiện *teleporttovillage* trong ví dụ trên, có 2 hành động được thực hiện trong **postactions** đó là cập nhật biến điều kiện *myhero\_go\_mapadventure-teleport-mapvillage* và thay đổi map hiện tại sang một map khác.

### 6.1.2. Kiến trúc module xử lý kịch bản

Để thực thi những thao tác đặc trưng khi một sự kiện được kích hoạt như nói một câu, hỏi một câu hỏi, chuyển qua map chiến đấu, cập nhật thuộc tính nhân vật... cần có một module riêng xử lý cho từng hành động đó ví dụ như PostActionUpdateAttribute, PostActionChangeMap, PostActionUpdateCondition. Trong kiến trúc này các lớp tương ứng với từng hành động sẽ kế thừa từ lớp cha PostAction

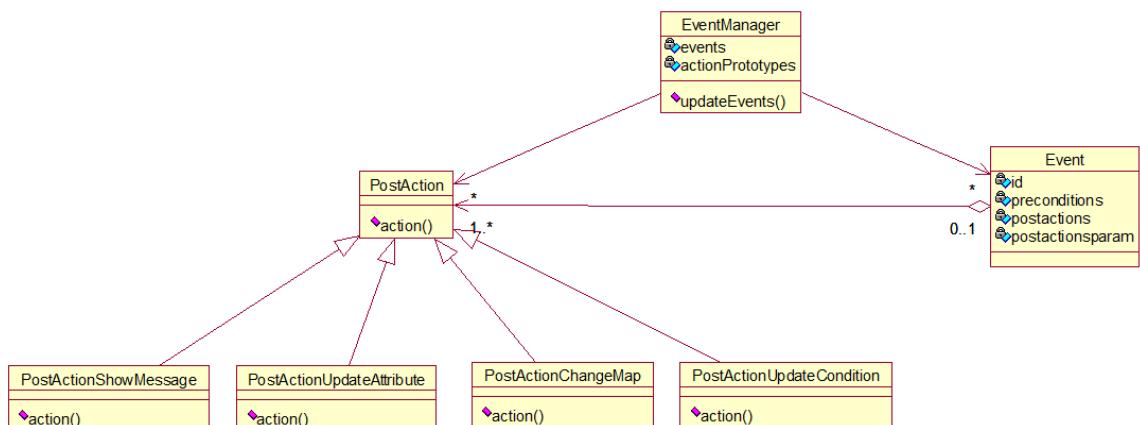
```
function PostAction() {
}
PostAction.prototype.action = function(parameter) { }
```

- **Action:** Hành động được thực thi khi sự kiện xảy ra. Nhận vào một chuỗi xml tham số. Chuỗi tham số này chính là chuỗi XML của node Parameter trong từng node PostAction của Event. Chuỗi tham số này sẽ được lớp PostAction cụ thể parse và lấy tham số để xử lý.



**Hình 6-1 Sơ đồ các lớp PostAction thực thi hành động của Event**

Chúng em cài đặt một lớp **EventManager** làm nhiệm vụ điều khiển việc xử lý kịch bản. **EventManager** sẽ đọc danh sách events từ file và lưu trữ danh sách các events có trong kịch bản. Ngoài ra lớp **EventManager** còn chứa một danh sách các PostAction được hỗ trợ. Trong EventManager có phương thức **updateEvents** để cập nhật lại tất cả các event được quản lý trong **EventManager** mỗi khi các biến điều kiện global thay đổi giá trị. Phương thức **updateEvents** sẽ duyệt qua từng event để kiểm tra nếu precondition của một event được thỏa thì thực hiện hành động postactions của event đó.



**Hình 6-2 Sơ đồ lớp module quản lý kịch bản**

#### ❖ Kết luận:

Với kiến trúc được thiết kế như trên có thể đáp ứng được việc hiện thực hóa cho khá nhiều kịch bản khác nhau và dễ dàng mở rộng cũng như chỉnh sửa lại kịch bản mà không cần can thiệp vào mã nguồn chương trình. Giải quyết được vấn đề được đặt ra về tính tiện dụng và linh động trong kịch bản của một engine game.

## 6.2. Kiến trúc quản lý màn chơi

### Vấn đề

Một màn chơi trong game được cấu tạo từ 3 thành phần, thứ nhất là địa hình, thứ 2 là các building trong màn chơi và cuối cùng là các nhân vật do hệ thống quản lý được đặt trên địa hình. Phần này trình bày về kiến trúc chúng em cài đặt để cho phép một màn chơi có thể được quy định từ file đặc tả bên ngoài, module xử lý của ứng dụng chỉ cần đọc file đặc tả này và xây dựng một màn chơi như mong muốn.

#### 6.2.1. Quản lý Building

Mỗi building sẽ bao gồm những thành phần sau:

- **id**: Id của building
- **typeid** : Loại building
- **model**: Mô hình building
- **scale, rotate, translate** : Các phép biến đổi trên building
- **attributeList**: Danh sách thuộc tính có thể có của building (xem 6.3)
- **load**: Load building
- **clone**: Hàm trả về bản sao của building
- **draw**: Hàm vẽ building

Chúng em cài đặt lớp BuildingManager quản lý tất cả các building được sử dụng trong game (xem Hình 6-3) . BuildingManager sẽ lưu trữ một danh sách các building mẫu (prototype) và trả về bản sao (clone) của building khi được yêu cầu.

Thành phần của BuildingManager bao gồm

- **buildingPrototypes**: Danh sách các building mẫu
- **getBuildingByID**: Hàm trả về bản sao của building

Danh sách các building mẫu được load từ file đặc tả xml từ bên ngoài. Cấu trúc file xml đặc tả cho building như sau:

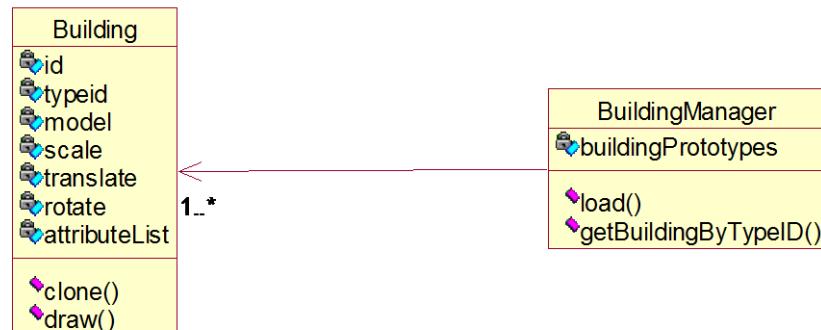
```
<buildings>
  <building>
    <typeid>humanbarrack</typeid>
    <modelid>humanbarrack</modelid>
    <textureid>humanbarrack</textureid>
    <effectid>basiceffect</effectid>
    <scale x="300" y="300" z="300"/>
    <translate x="0" y="0" z="0"/>
```

```

<rotate x="0" y="0" z="0"/>
<attributes></attributes>
</building>
<building>
    <typeid>humantower</typeid>
    <modelid>humantower</modelid>
    <textureid>humantower</textureid>
    <effectid>basiceffect</effectid>
    <scale x="200" y="200" z="200"/>
    <translate x="0" y="0" z="0"/>
    <rotate x="0" y="0" z="0"/>
    <attributes></attributes>
</building>
</buildings>

```

Trong file đặc tả building, ta cần các thông tin *modelid*, *textureid*, *effectid* để khởi tạo thuộc tính *model* trong lớp Building.



**Hình 6-3 Quản lý building**

### 6.2.2. Quản lý Character

Một đối tượng Character sẽ bao gồm các thành phần sau :

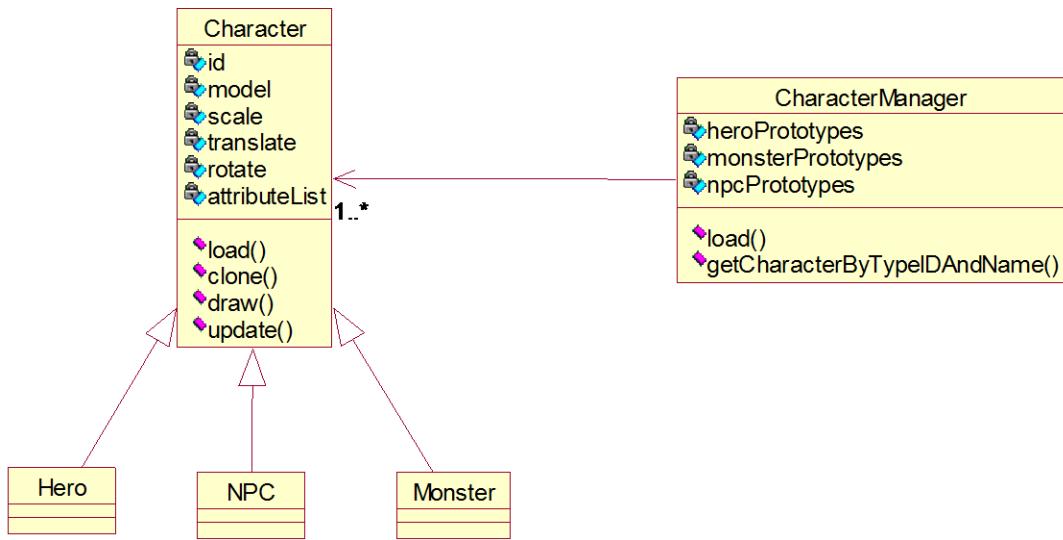
- **id:** Id của nhân vật
- **model:** Mô hình nhân vật
- **scale, rotate, translate :** Các phép biến đổi của nhân vật
- **attributeList:** Danh sách thuộc tính có thể có của nhân vật (xem 6.3)
- **load:** Load nhân vật
- **clone:** Trả về bản sao của nhân vật
- **draw:** Vẽ nhân vật
- **update:** Cập nhật nhân vật

Tương tự như Building, để quản lý Character trong game nhóm em cài đặt lớp CharacterManager (xem Hình 6-4). CharacterManager có nhiệm vụ lưu trữ danh sách tất cả các Character sử dụng trong game được load từ một file đặc tả xml bên ngoài và trả về bản sao của Character khi được yêu cầu. Các thành phần của CharacterManager bao gồm:

- **heroPrototype** : Danh sách Character loại Hero
- **monsterPrototype** : Danh sách Character loại Monster
- **npcPrototype** : Danh sách Character loại NPC
- **getCharacterByIDAndName** : Trả về bản sao của Character

Danh sách các Character mẫu được load từ file đặc tả xml từ bên ngoài. Cấu trúc file xml đặc tả cho Character như sau:

```
<characters>
    <character>
        <typeid>hero</typeid>
        <name>lichking</name>
        <attributes>
            <attribute typeid="hp" curval="1000"/>
            <attribute typeid="mana" curval="500"/>
            <attribute typeid="damage" curval="10"/>
            <attribute typeid="armor" curval="5"/>
        </attributes>
        <modelid>lichkingbody</modelid>
        <textureid>lichkingbody</textureid>
        <effectid>basiceffect</effectid>
        <scale x="1" y="1" z="1"/>
        <translate x="0" y="0" z="0"/>
        <rotate x="0" y="0" z="0"/>
    </character>
    <character>
        <typeid>monster</typeid>
        <name>pitlord</name>
        <attributes>
            <attribute typeid="hp" curval="1000"/>
            <attribute typeid="damage" curval="5"/>
            <attribute typeid="armor" curval="1"/>
        </attributes>
        <modelid>pitlord</modelid>
        <textureid>pitlord</textureid>
        <effectid>basiceffect</effectid>
        <scale x="1" y="1" z="1"/>
        <translate x="0" y="0" z="0"/>
        <rotate x="0" y="0" z="0"/>
    </character>
</characters>
```



**Hình 6-4 Quản lý Character**

### 6.2.3. Quản lý Map

Lớp đối tượng Map sẽ bao gồm các thành phần sau:

- **terrain** : Địa hình của Map
- **skybox** : Đối tượng Skybox
- **buildings** : Danh sách Building
- **characters** : Danh sách Character
- **load** : Load Map
- **draw** : Vẽ Map
- **update** : Cập nhật Map
- **getHeight**: Trả về độ cao tại một vị trí trên Map

Nhóm em đã cài đặt lớp MapManager để quản lý danh sách tất cả Map trong game (xem Hình 6-5). MapManager sẽ load danh sách tất cả Map từ một file xml bên ngoài, mỗi Map sẽ có một id xác định và một file xml cấu hình riêng.

```

<maps>
    <map id="mapvillage" xmlid="mapvillage"/>
    <map id="mapadventure" xmlid="mapadventure"/>
</maps>

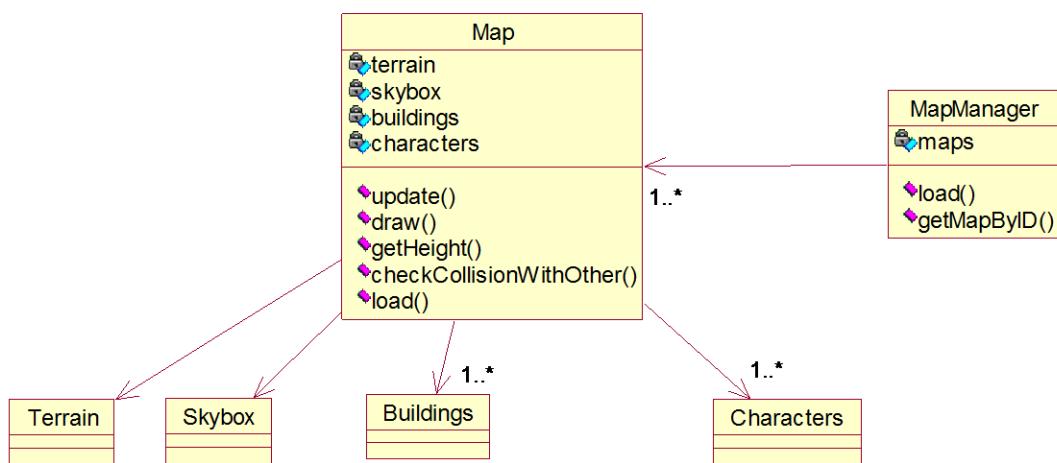
```

File xml cấu hình Map sẽ có cấu trúc như sau:

```

<map>
    <characters xmlid="mapcharacters"/>
    <skybox>
        <effectid>skyboxeffect</effectid>
        <back>back</back>
        <front>front</front>
        <left>left</left>
        <right>right</right>
        <top>top</top>
        <bottom>grassbottom</bottom>
    </skybox>
    <terrain>
        <width>256</width>
        <height>256</height>
        <heightmapid>terrainingvillage</heightmapid>
        <blockscale>36</blockscale>
        <heightscale>3</heightscale>
        <effectid>terraineffect</effectid>
        <heightlevel min="0" max="0" textureid="water"/>
        <heightlevel min="-1" max="50" textureid="grass"/>
        <heightlevel min="100" max="150" textureid="highland"/>
        <heightlevel min="200" max="255" textureid="snow"/>
    </terrain>
    <buildings>
        <building>
            <id>mapvillage-humanlumbermill</id>
            <typeid>humanlumbermill</typeid>
            <column>120</column>
            <row>60</row>
            <scale x="1" y="1" z="1"/>
            <rotate x="0" y="45" z="0"/>
        </building>
    <buildings>
</map>

```



Hình 6-5 Quản lý Map

## ❖ Kết luận

Với kiến trúc thành phần quản lý màn chơi được cài đặt có thể đáp ứng được nhu cầu thiết kế một màn chơi mới cũng như chỉnh sửa một màn chơi có sẵn một cách dễ dàng. Việc này cung cấp một cơ chế cho phép tạo dựng một màn chơi mới thậm chí một game mới với cốt truyện, địa hình, khung cảnh, kiến trúc, con người hoàn toàn mới rất nhanh chóng mà không cần phải lập trình lại một game mới. Việc duy nhất cần làm đó là đầu tư đồ họa mô hình, giao diện đồng thời thiết kế một cốt truyện thật hay và hấp dẫn, sau đó truyền các kịch bản được thiết kế và các tài nguyên đồ họa này vào cho ứng dụng lập tức sẽ có ngay một game như ý muốn. Việc này tiết kiệm thời gian và chi phí rất nhiều về mảng lập trình game khi phát triển một game mới cùng dòng game và đây cũng là ưu điểm nổi bật của game engine được chúng em cài đặt.

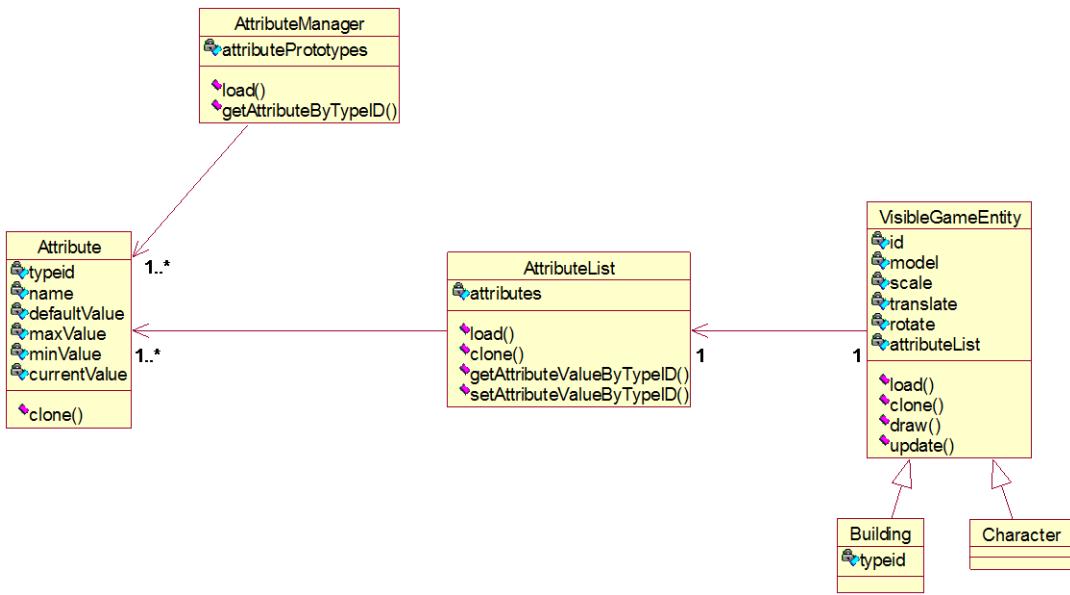
### 6.3. Kiến trúc quản lý thuộc tính trong game

#### Vấn đề

Các nhân vật trong game thường có một số thuộc tính như máu, năng lượng, kinh nghiệm, sức đánh, phòng thủ ... Vấn đề đặt ra là làm thế nào quản lý cá thuộc tính này đồng thời cho phép mở rộng danh sách thuộc tính.

#### Giải pháp

Để giải quyết được vấn đề này nhóm em đã tạo ra ba lớp đó là *Attribute*, *AttributeList* và *AttributeManager*. Mỗi đối tượng trong game sẽ có một thuộc tính kiểu *AttributeList* để quản lý danh sách các *Attribute*. *AttributeManager* có nhiệm vụ load từ file xml và lưu trữ danh sách các *Attribute* mẫu (prototype) trong chương trình có sử dụng.



**Hình 6-6 Quản lý thuộc tính**

Cấu trúc của file xml liệt kê tất cả thuộc tính sử dụng trong game được nhóm em đề xuất như sau:

```

<attributes>
    <attribute>
        <typeid>hp</typeid>
        <name>Hit point</name>
        <defval>1000</defval>
        <maxval>10000</maxval>
        <minval>0</minval>
        <curval>1000</curval>
    </attribute>
    <attribute>
        <typeid>mp</typeid>
        <name>Mana point</name>
        <defval>500</defval>
        <maxval>5000</maxval>
        <minval>0</minval>
        <curval>500</curval>
    </attribute>
    <attribute>
        <typeid>damage</typeid>
        <name>Damage</name>
        <defval>100</defval>
        <maxval>1000</maxval>
        <minval>0</minval>
        <curval>100</curval>
    </attribute>
</attributes>

```

Dưới đây là ví dụ minh họa cách đặt tả thuộc tính của nhân vật.

```
<character>
    <typeid>herolichking</typeid>
    <name>lichking</name>
    <attributes>
        <attribute typeid="hp" curval="2000"/>
        <attribute typeid="mp" curval="1000"/>
        <attribute typeid="damage" curval="1000"/>
    </attributes>
</character>
```

## 6.4. Xử lý việc hiển thị lời thoại

### Vấn đề

Trong game, khi nhân vật thực hiện các hành vi cơ bản tác động đến các nhân vật khác như nói chuyện, nhận – trả nhiệm vụ... đều cần đến lời thoại. Lời thoại là một phương pháp dẫn dắt, trao đổi thông tin giữa người chơi và chương trình. Các lời thoại hay, hấp dẫn sẽ làm cho game thêm phong phú và lôi cuốn hơn. Hơn nữa đối với một kịch bản game nhập vai thì lời thoại là một phần không thể thiếu. Vấn đề là hiện thị lời thoại đó như thế nào để người chơi dễ đọc và theo sát được tiến trình trong game.

### Giải pháp

Giải pháp mà nhóm em áp dụng là vẽ một khung nồi trong thế giới 3 chiều để hiển thị lời thoại (ví dụ như trên đầu nhân vật đang nói). Đây là một giải pháp phức tạp hơn vì liên quan tới nhiều yếu tố hơn như toạ độ thế giới thực, góc nhìn, ... Với cách hiển thị này sẽ hơi khó đọc cho người chơi nên chỉ thích hợp để biểu diễn trạng thái, thái độ của đối tượng nhân vật, như một câu nói chung chung, hay những dẫu châm than, châm hỏi trên đầu nhân vật, ... Để vẽ được một hộp thoại lên không gian 3D như vậy ta cần làm 2 bước sau:

- **Bước 1:** Render toàn bộ khung hiển thị cùng nội dung hội thoại lên một ảnh sử dụng canvas.
- **Bước 2:** Dùng kỹ thuật Billboarding để vẽ ảnh có được ở bước 1 lên một toạ độ trên thế giới 3D và texture này luôn xoay đối mặt với camera để người chơi có thể nhìn thấy nội dung hội thoại.



**Hình 6-7 Xử lý lời thoại trong game**

## **6.5. Kết luận**

Nội dung chương này trình bày các vấn đề liên quan đến kiến trúc của game. Với kiến trúc này cho phép chúng ta tạo ra các màn chơi theo các thông số cho trước, thay đổi các thuộc tính, số lượng các thành phần, nhân vật trong game. Ngoài ra ta còn có thể thay đổi kịch bản từ các thông số được lưu trữ trong các file dữ liệu bên ngoài.

## Chương 7

### Kết luận

 Nội dung của chương này trình bày các kết quả đạt được của các ứng dụng đã xây dựng trong đề tài và hướng phát triển của đề tài.

#### 7.1. Các kết quả đạt được

Sau quá trình tìm hiểu và thực nghiệm, chúng em đã xây dựng thành công **Game Framework 3D** bằng công nghệ **WebGL** và sử dụng Framework này xây dựng được 2 game với 2 thể loại game khác nhau. Chúng em đặt tên cho 2 game lần lượt là **Devil** (nhập vai), **GunPlay** (hành động).

Các chức năng chính của **Framework** :

- Cho phép người lập trình có thể cấu hình địa hình của màn chơi, thay đổi số lượng, loại character, building, NPC tùy ý bằng cách thay đổi thông tin thuộc tính trong file cấu hình XML.
- Ngoài ra còn có thể thay đổi kịch bản game một cách dễ dàng bằng cách thay đổi các thông tin trong file XML EventList.

Xây dựng được game **Devil** (nhập vai) :

- Animation hài hòa như thật.
- Camera thông minh, tự động nằm phía sau nhân vật khi di chuyển
- Thiết lập địa hình lòi lõm đồi núi
- Di chuyển trên địa hình đồi núi
- Tích hợp kỹ thuật Billboard : cột máu của các con quái để nhìn ở góc độ nào cũng có thể thấy được đầy đủ cột máu đó.
- Hiệu ứng Particle : hào quang của nhân vật khi chọn phép hào quang
- Xử lý va chạm : khi nhân vật đụng object building thì không thể đi về phía trước được, mà trượt theo bề mặt object building.
- RayPicking : chưởng phép đến vị trí click chuột trên địa hình đồi núi
- Menu 2D : sử dụng các thẻ trong HTML để thiết lập giao diện Menu
- Sound : nhạc nền, nhạc khi chưởng phép, ...

- Multiplayer : ngoài việc chơi SinglePlayer, game của chúng em tích hợp Multiplayer để các người chơi có thể chơi chung với nhau trong một phòng chơi.
- Chat : các người chơi chung một phòng chơi có thể chat nói chuyện với nhau.

Xây dựng được game **GunPlay** (hành động) :

- Animation hài hòa, thật.
- Camera được đặt ở vị trí mắt của nhân vật, quay nhân vật, chỉnh tọa độ súng một cách dễ dàng.
- Thiết lập địa hình lồi lõm đồi núi
- Di chuyển trên địa hình đồi núi
- Tích hợp kỹ thuật Billboard : cột máu của các con quái để nhìn ở góc độ nào cũng có thể thấy được đầy đủ cột máu đó.
- Xử lý va chạm : khi nhân vật đụng object building thì không thể đi về phía trước được, mà trượt theo bề mặt object building.
- Bắn súng : khi người chơi click chuột thì nhân vật sẽ bắn súng về hướng mà camera hướng đến.
- Menu 2D : sử dụng các thẻ trong HTML để thiết lập giao diện Menu
- Sound : nhạc nền, nhạc khi bắn súng, ...
- Multiplayer : ngoài việc chơi SinglePlayer, game của chúng em tích hợp Multiplayer để các người chơi có thể chơi chung với nhau trong một phòng chơi.
- Chat : các người chơi chung một phòng chơi có thể chat nói chuyện với nhau

Người chơi có thể kết nối chơi với nhau qua mạng LAN

Game được xây dựng bằng công nghệ WebGL, nên ngoài việc hiển thị game trên các trình duyệt hỗ trợ WebGL trên các hệ điều hành của máy tính để bàn hay laptop như Linux, Window, Mac. Game có thể hiển thị trên các thiết bị di động như : Nokia N900, Iphone4/iPhone4S/iPad2/iPad3, các thiết bị Android 1.6+ sử dụng Opera Mobile 12.



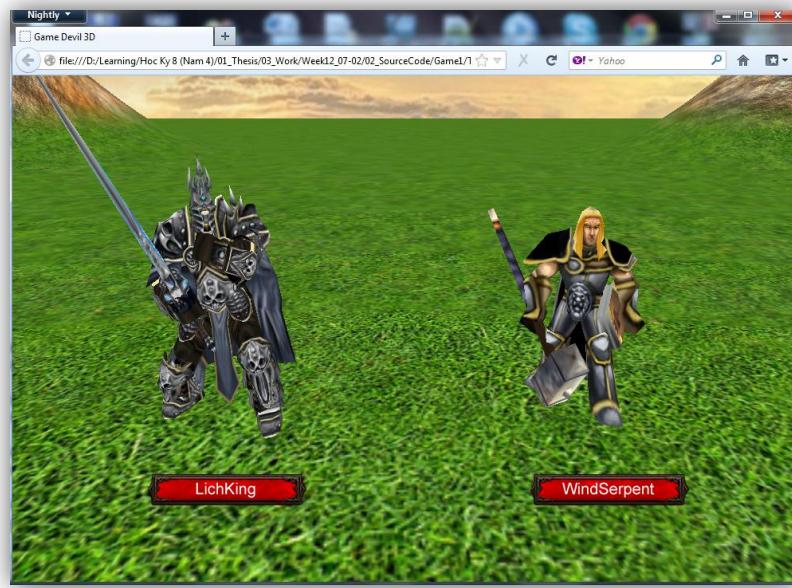
**Hình 7-1 Chơi game GunPlay bằng điện thoại LG-GT540 Android 2.3**

Game nhập vai hay hành động được xây dựng cho máy tính để bàn hay laptop thì khi hiển thị trên các thiết bị di động, chỉ có thể chơi được các chức năng bằng cách touch vào màn hình thay cho sự kiện click chuột, nhưng không thể di chuyển được vì game được thiết kế di chuyển bằng cách input bằng keyboard, mà thiết bị di động không có keyboard device. Vì thế, nhóm chúng em đã tạo thêm một phiên bản game giành cho Mobile, cho phép người dùng di chuyển, đánh quái, chưởng, bắn, thay đổi góc nhìn với các thao tác hoàn toàn trên màn hình như game Ironman trên Iphone/Ipad.

## **7.2. Một số hình ảnh minh họa trong game**

### **7.2.1. Game Devil :**

Sau khi load dữ liệu xong, game hỗ trợ chọn nhân vật, ở đây có 2 loại nhân vật : LichKing, WindSerpent



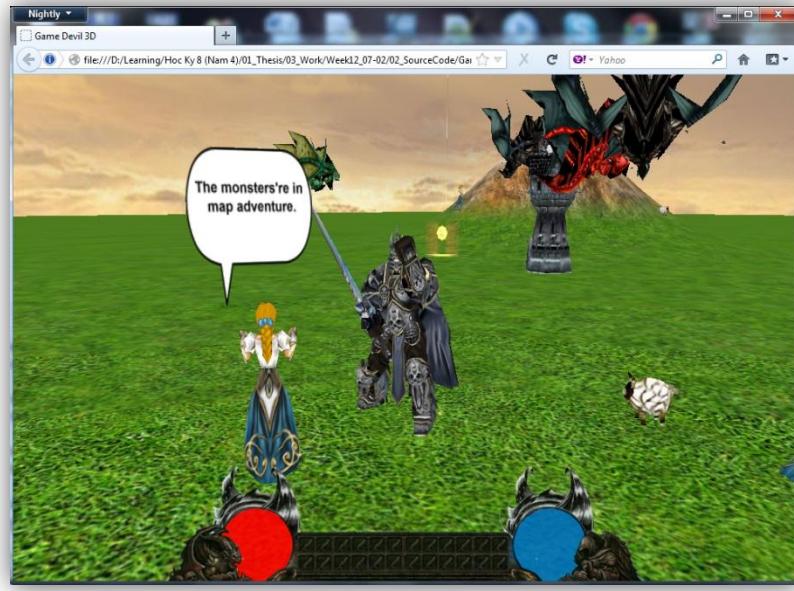
**Hình 7-2 Màn hình chọn nhân vật**

Bắt đầu chơi game



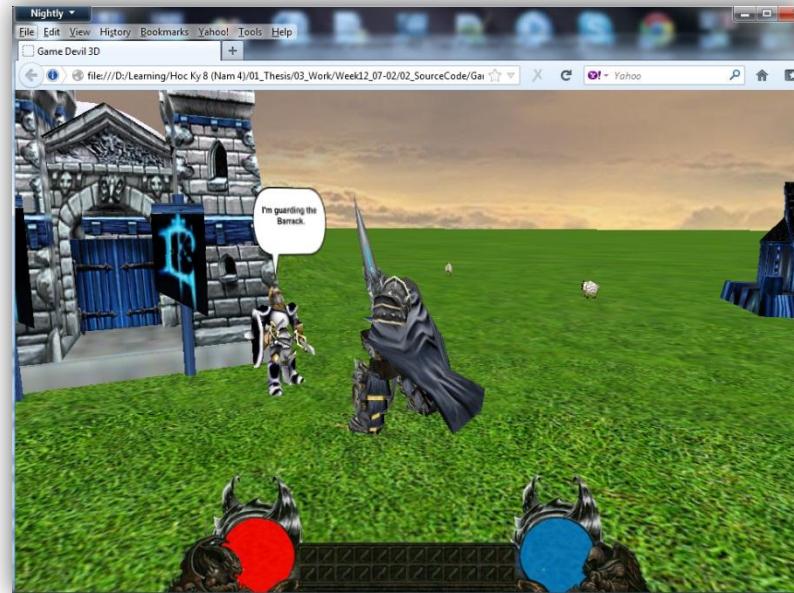
**Hình 7-3 Màn hình bắt đầu chơi game**

Chúng ta đang ở thế giới của nhân vật, ở thế giới này, nhân vật sẽ nhận được lời hướng dẫn của người chỉ đường, sang thế giới devil đánh quái.



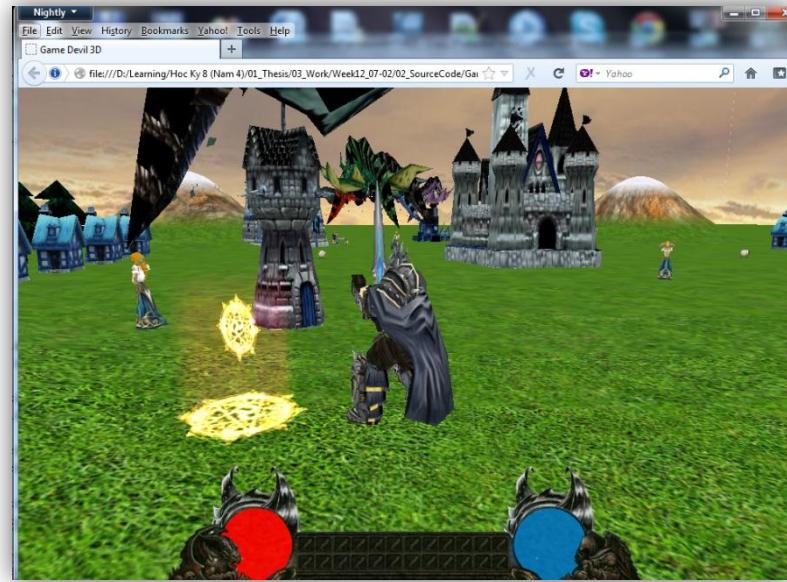
**Hình 7-4 Người chỉ đường hướng dẫn nhân vật sang thế giới Devil**

Chúng ta có thể trò chuyện với các NPC trong thế giới của nhân vật



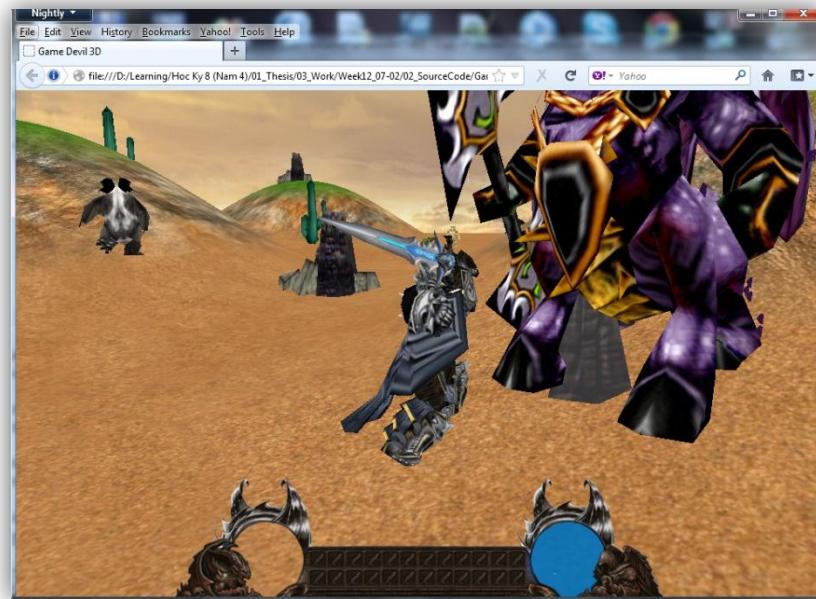
**Hình 7-5 Trò chuyện với NPC**

Để chuyển sang thế giới Devil, chúng ta phải dùng vào Teleport



**Hình 7-6 Sang thế giới Devil bằng Teleport**

Sau khi sang thế giới Devil, có rất nhiều quái vật, chúng ta sẽ phải đánh thắng Devil không lồ, mới được quay về thế giới của nhân vật.



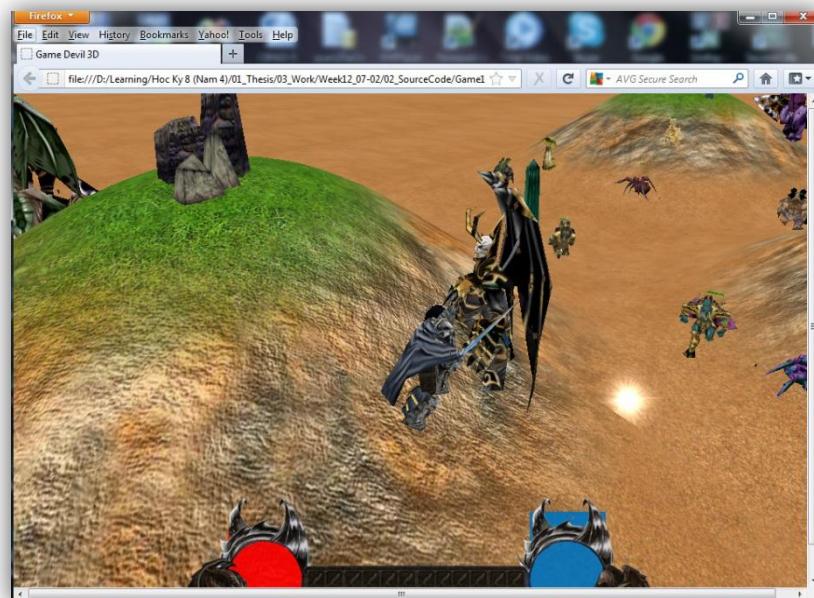
**Hình 7-7 Cảnh đánh quái vật ở thế giới Devil**

Ngoài chức năng đánh gần bằng kiếm, chúng ta có thể tấn công quái vật bằng các chiêu chưởng



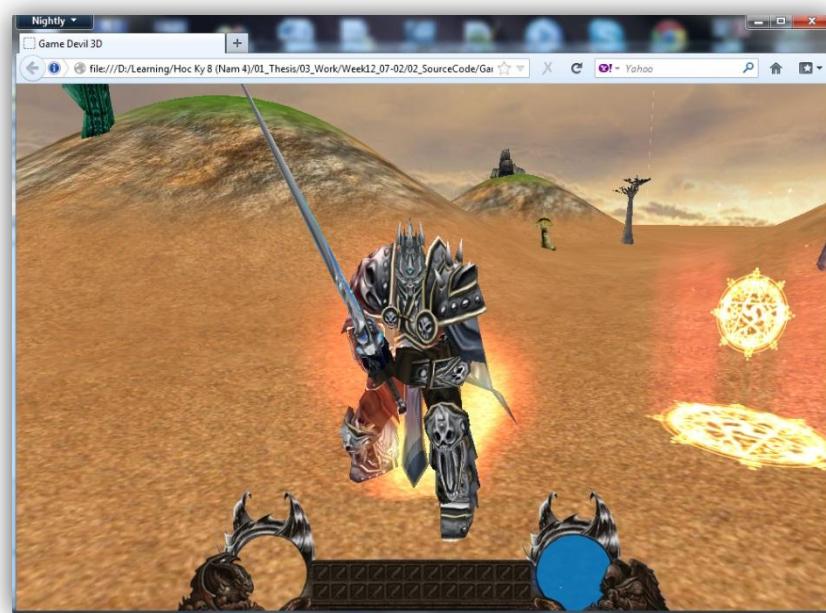
**Hình 7-8 Bảng thông tin nhân vật**

Khi chúng ta chọn chiêu chưởng bằng cách chọn 1 trong 3 chiêu ở bảng nhân vật, chúng ta có thể chưởng bằng cách click chuột



### **Hình 7-9 Chưởng phép bằng cách click chuột lên màn hình**

Ngoài chiêu chưởng ra, để làm tăng sức đánh kiếm, chúng ta có thể chọn chiêu thứ 3 trong bảng nhân vật



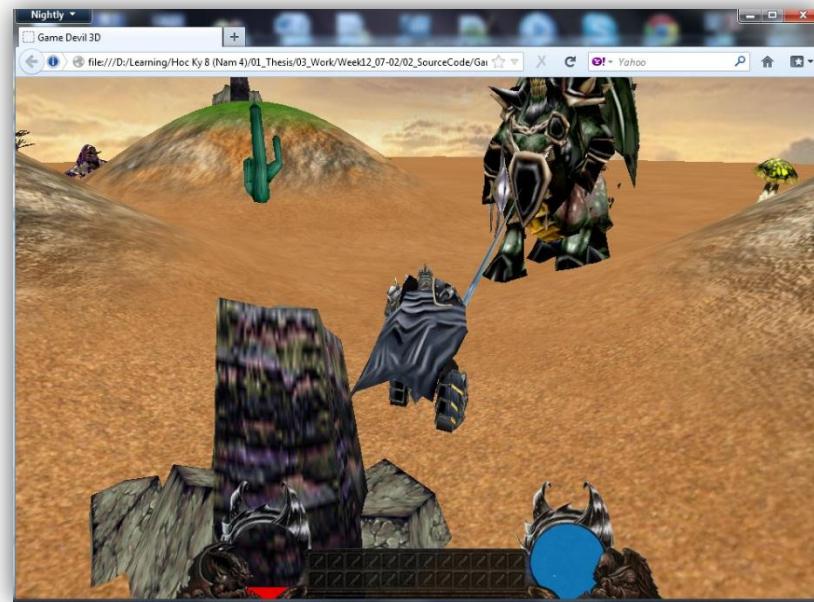
### **Hình 7-10 Chiêu thứ 3 : hào quang xuất hiện quanh nhân vật**

Chúng ta có thể xoay camera quanh nhân vật, và có thể thấy được toàn cảnh thế giới



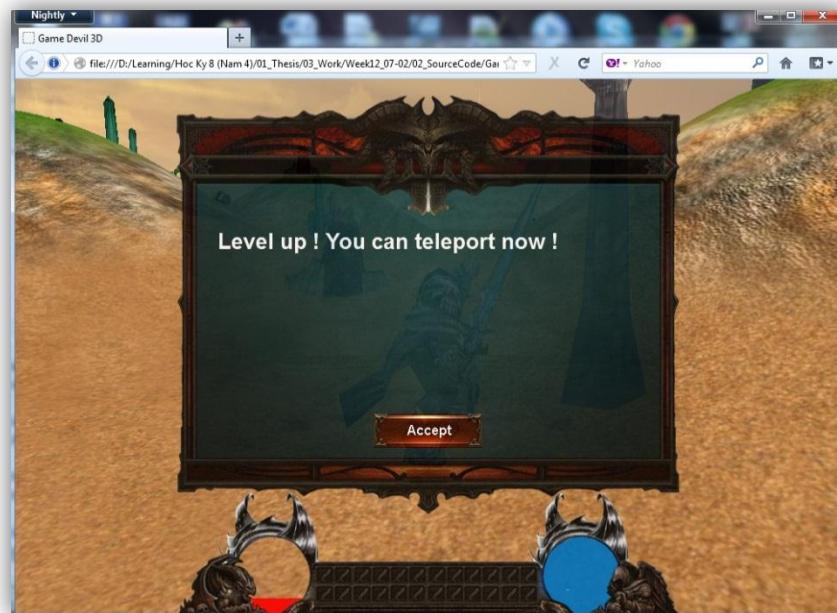
**Hình 7-11 Nhìn ở góc nhìn khác**

Để quay về thế giới của nhân vật, chúng ta phải đánh bài Devil khổng lồ



**Hình 7-12 Chuẩn bị đấu với Devil khổng lồ**

Sau khi đánh thắng Devil khổng lồ, chúng ta nhận được thông báo tăng level và có thể quay về thế giới nhân vật qua Teleport



**Hình 7-13 Hiện thông báo khi đánh thắng Devil không lồ**

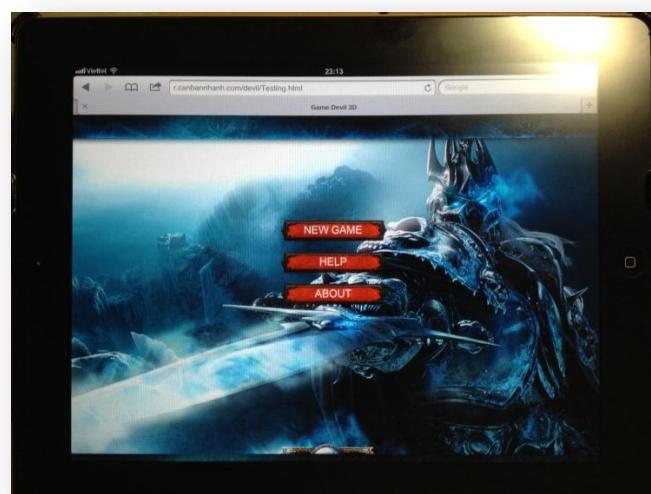
Chơi game Devil ở chế độ Multiplayer, người dùng có thể thấy nhau, trò chuyện với nhau

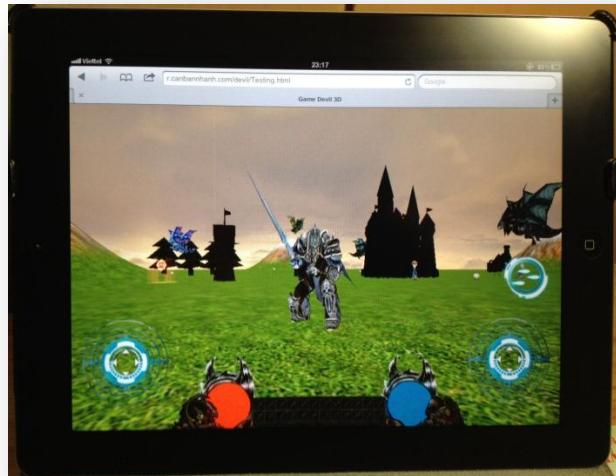
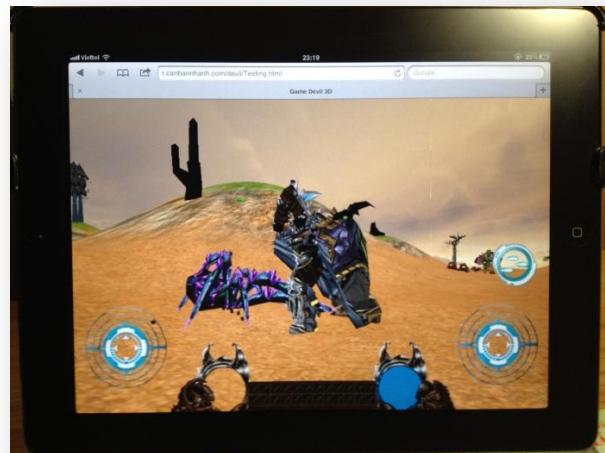
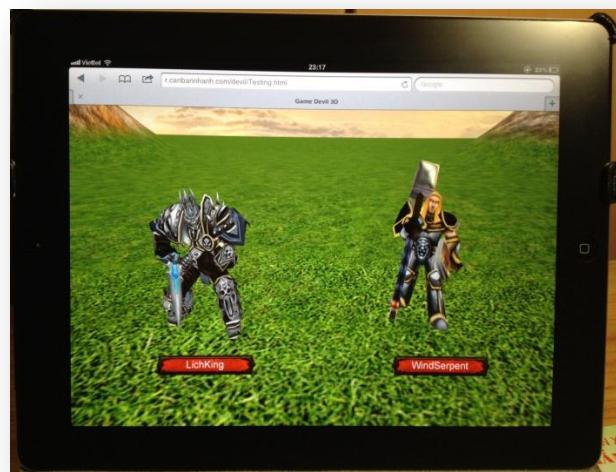


**Hình 7-14 Chơi ở chế độ Multiplayer**

### **7.2.2. Game Devil phiên bản Mobile :**

Menu Game





### **7.2.3. Game GunPlay :**

Đầu tiên, thể hiện một số hình minh họa khi chơi ở chế độ SinglePlayer, người chơi có thể chọn phe của mình : khủng bố hoặc chống khủng bố.



**Hình 7-15 Màn hình Menu chọn phe**

Bắt đầu vào chơi game



**Hình 7-16 Bắt đầu vào cảnh chơi**

Di chuyển và quan sát quang cảnh khi đứng trên núi



**Hình 7-17 Quan sát khung cảnh với góc nhìn đứng trên đồi núi**

Hai phe khi cách nhau một khoảng nhất định, sẽ phát hiện ra nhau và tấn công lẫn nhau bằng cách bắn đối phương



**Hình 7-18 Hai phe bắn nhau**

Đối phương phát hiện ra mình và tấn công nhân vật của mình



**Hình 7-19 Đối phương phát hiện và bắn nhân vật của mình**

Cảnh một nhân viên chống khủng bố bị phe khủng bố bắn chết



**Hình 7-20 Một quân bị chết**

Khi nhân vật của mình bị đối phương bắn hết máu, sẽ hiện màn hình Menu thông báo và kết thúc màn chơi



**Hình 7-21 Màn hình Menu khi bị bắn hết máu**

Chơi một màn chơi khác



**Hình 7-22 Thay đổi màn chơi sang Map khác**

Chơi theo chế độ Multiplayer



**Hình 7-23 Hai người chơi thấy nhau trong một màn chơi**

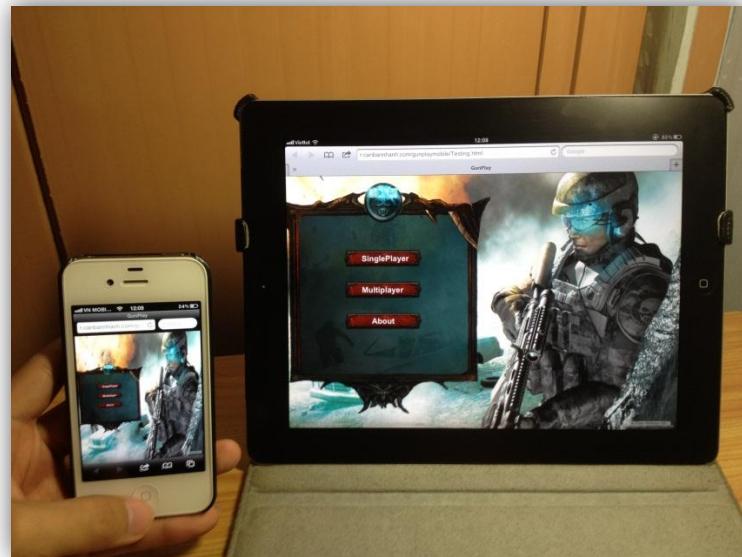
Người chơi chơi cùng một phòng có thể chat với nhau



**Hình 7-24 Chat với nhau trong Game ở chế độ Multiplayer**

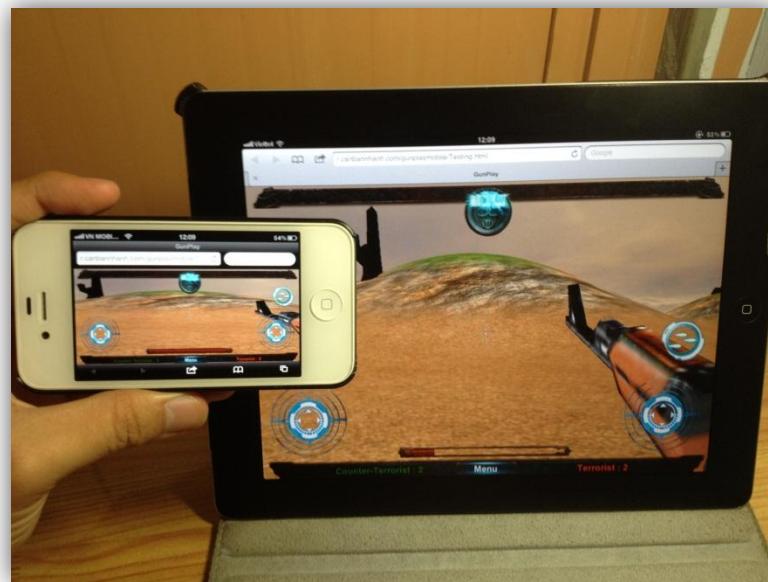
#### **7.2.4. Game GunPlay phiên bản Mobile :**

Chơi game GunPlay bằng Iphone4S và Ipad2



**Hình 7-25 Menu chính của Game**

Màn hình khi chơi game



**Hình 7-26 Chơi game ở chế độ SinglePlayer**

Người chơi có thể di chuyển, thay đổi tầm nhìn và bắn đạn bằng cách touch vào 3 hình màu xanh tương ứng trên màn hình.

### **7.3. Hướng phát triển của đề tài**

- Xử lý AI cho nhân vật được thông minh hơn.
- Thiết kế game Editor, cho phép người dùng tạo dựng địa hình tùy ý.
- Thiết kế thêm phòng bán vũ khí, vật phẩm để người chơi có thể mua bán.
- Làm thêm nhiều chiêu chưởng Particle đẹp mắt khác.
- Mở rộng game với những ý tưởng mới.

## TÀI LIỆU THAM KHẢO

- [1] Khronos Group, <https://www.khronos.org/registry/webgl/specs/1.0/>
- [2] Learning WebGL, [http://learningwebgl.com/blog/?page\\_id=1217](http://learningwebgl.com/blog/?page_id=1217)
- [3] Aaron Reed, *Learning XNA 3.0: XNA 3.0 Game Development for the PC, Xbox 360, and Zune.*: O'Reilly Media, 2008.
- [4] W3Schools, Refsnes Data, 1999 – 2011, <http://www.w3schools.com>
- [5] OpenGL Reference Manual <http://glprogramming.com/blue/>
- [6] OpenGL Programming Guide. <http://glprogramming.com/red/>
- [7] Web goes 3D – Does Advertising too? Won don Technical Lead Domus Inc. 29-04-2011 . <http://www.codeproject.com/Articles/188499/Web-goes-3D-does-advertising-too-The-WebGL-Silverl>
- [8] Hướng dẫn lập trình OpenGL Cơ Bản - Đặng Nguyễn Đức Tiến – Vũ Quốc Hoàng - Lê Phong
- [9] Pro HTML5 Programming, Peter Lubbers, Brian Albers, Frank Salim, 2010, Apress.

## PHỤ LỤC

### A. Các vấn đề về Resource 3D :

WebGL có thể load được các dạng file : \*.js, \*.json, \*.dae, \*.obj, ... Nhưng để đơn giản và dễ quản lý, nhóm chúng tôi đã chọn cách load model từ file \*.json.

Có nhiều cách để lấy resource model 3D của các game khác nhau. Ở đây, nhóm chúng em xin trình bày cách lấy resource model 3D trong game Warcraft 3 và cách chuyển đổi sang file \*.json

#### ❖ Lấy resource từ các diễn đàn Warcraft 3:

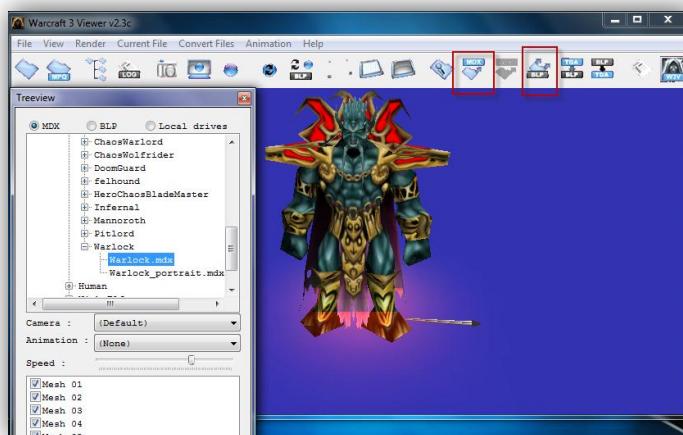
Warcraft Campaigns : <http://www.wc3c.net/resources.php?f=640>

HiveWorkShop : <http://www.hiveworkshop.com/forums/models.php>

#### ❖ Lấy resource bằng phần mềm “Warcraft 3 Viewer”

Link download :

[http://dl.dropbox.com/u/25839398/Thesis/04\\_Software/Warcraft3Viewer/Warcraft3Viewer.zip](http://dl.dropbox.com/u/25839398/Thesis/04_Software/Warcraft3Viewer/Warcraft3Viewer.zip)



Phần mềm hack resource game Warcraft III - “Warcraft 3 Viewer”

Cách sử dụng phần mềm Warcraft 3 Viewer :

- Sau khi đã cài đặt Warcraft 3. Warcraft 3 Viewer sẽ tự động tìm đến file war.mpq để view model 3D

- Chọn Model 3D ở thanh bar bên trái.
  - o Extract \*.mdx : file model 3d
  - o Extract \*.BLP : file texture (\*.bmp, \*.jpg, \*.tga)

#### ❖ Hướng dẫn Export sang \*.json :

##### ⊕ Tổng quát :

Khi lấy model từ Warcraft III : file model có dạng \*.mdx, file texture có dạng \*.jpg, \*.bmp, \*.tag, \*.blp. Vì thế, việc của chúng ta bây giờ là chuyển từ dạng \*.mdx sang dạng \*.json. Cách tổng quát như sau :

- Import \*.mdx vào 3DS MAX
- Chọn Mesh và Export sang \*.obj từ 3DS MAX
- Import \*.obj vào Blender 2.49a
- Từ Blender 2.49a Export sang WebGL.js

Đây là cách lấy được một Frame, để lấy được bộ Vertices hay bộ Animation, thì chúng ta phải làm lại thao tác này nhiều lần, sau đó ghép lại thành 1 file \*.json mong muốn.

##### ⊕ Chuẩn bị :

Cài đặt các phần mềm hỗ trợ trong việc thiết kế model :

- 3DS Max 2009/2010/2011/2012
- Blender 2.49a

##### ⊕ Cài đặt và sử dụng 3DS MAX :

Sau khi cài đặt 3DS MAX 2009/2010/2011/2012. Chúng ta phải cài thêm plugin import file \*.mdx để có thể import được các file mình đã extract từ Warcraft 3 Viewer.

Link download plugin:

[http://dl.dropbox.com/u/25839398/Thesis/04\\_Software/3DSMax/Plugin/imexpmd\\_x\\_v2.0.3.zip](http://dl.dropbox.com/u/25839398/Thesis/04_Software/3DSMax/Plugin/imexpmd_x_v2.0.3.zip)

Copy file **impexpmdxv2.0.3** vào thư mục plugins của 3DS MAX. Sau đó restart 3DS MAX.

Cách Import file \*.mdx trong 3DS MAX :

- Click vào biểu tượng cây búa ở bên phải màn hình
- Chọn MAXScript
- Chọn MDX Import/Export ở mục Utilities
- Import \*.mdx



*Kết quả khi import thành công file \*.mdx*

Ở đây các model extract từ Warcraft 3 đều có animation (stand, walk, attack, victory...). Chúng ta chọn từng frame để extract ra file \*.obj. Muốn extract ra frame nào thì play đến frame đó và extract ra \*.obj từ thanh công cụ của 3DS MAX.

#### Cài đặt và sử dụng Blender :

Chúng ta nên sử dụng Blender 2.49a tính đến thời điểm 19/05/2012. Vì các plugin của các phiên bản Blender về sau này chưa được hoàn chỉnh, nên import và export chưa hết được những dữ liệu mình cần. VD : Blender 2.62 chạy rất tốt, nhưng không export ra WebGL.js. Mình sử dụng Blender 2.49a : vừa có thể export ra WebGL.js (vertices, normals, texCoord), vừa có thể import file \*.obj từ 3DS MAX đã export ra.

Link download plugin :

- WebGLExport.py :  
[http://dl.dropbox.com/u/25839398/Thesis/04\\_Software/Blender/Plugin/WebGL/249a/WebGLExport.py](http://dl.dropbox.com/u/25839398/Thesis/04_Software/Blender/Plugin/WebGL/249a/WebGLExport.py)
- Gói import obj :  
[http://dl.dropbox.com/u/25839398/Thesis/04\\_Software/Blender/Plugin/Obj/249a/Obj249a.rar](http://dl.dropbox.com/u/25839398/Thesis/04_Software/Blender/Plugin/Obj/249a/Obj249a.rar)

Cài đặt Plugin cho Blender 2.49a :

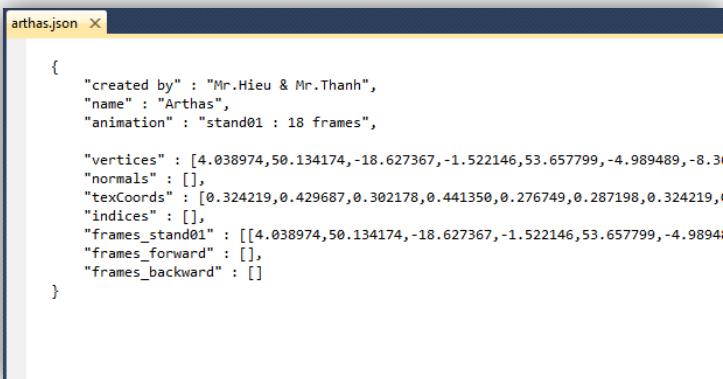
- Đối với phiên bản Blender 2.49a : chúng ta phải tạo ra thư mục chứa file plugin, chứ không được như Blender 2.62 là chúng ta chỉ cần copy vào thư mục đã được tạo sẵn bởi Blender.
- Tạo thư mục : “.blender” => “scripts” trong thư mục Blender ở ổ C mình đã cài đặt Blender.
- Copy 2 gói plugin vừa download về vào.
- Restart Blender 2.49a.

Giờ đây chúng ta có thể Import được file \*.obj đã export từ 3DS MAX và từ đó có thể export ra file \*.js như mình mong đợi. Nhưng mỗi lần export ra \*.js chúng ta chỉ có Vertices, Normal, TexCoord.

Vì thế, chúng ta muốn có được 1 bộ Animation thì phải làm lại thao tác này nhiều lần để có được bộ vertices, sử dụng nó để làm Animation :

- Export từng frame từ 3DS MAX sang \*.obj
- Import \*.obj vào Blender 2.49a
- Export ra \*.js

Từ đó chúng ta có thể tạo ra file JSON mong muốn bao gồm : Vertices, Normal, Frames.



## Cấu trúc model 3D file \*.json

Import \*.obj trong Blender 2.49a : Click vào button “-X90” để nó không quay theo chiều X 90 độ nữa. Vì trong code mình không cần. Sau đó nhấn Import.

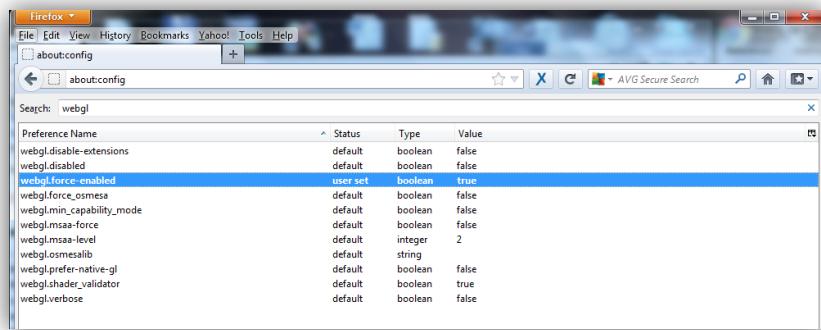
Export sang \*.js trong Blender 2.49a : không cần chọn button gì hết, chỉ cần chọn đường dẫn export. Sau đó nhấn Export.

#### B. Các trình duyệt và nền tảng hỗ trợ WebGL :

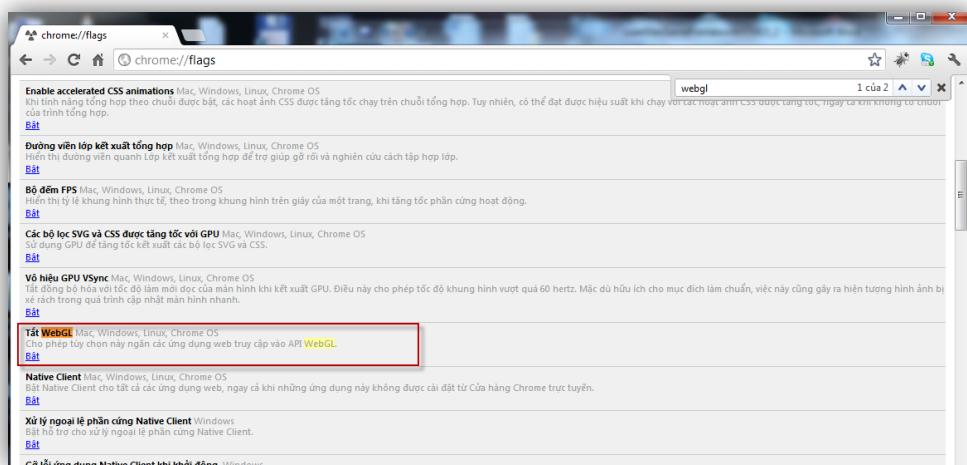
Đến thời điểm 07/2012, đã có nhiều trình duyệt trên Desktop Browser hỗ trợ WebGL : Mozilla Firefox 4+, Google Chrome 9+, Safari 5.1+, Opera 11+, Internet Explorer 6+ (phải cài plugin Chrome Frame và IEWebGL). Về mảng thiết bị di động, một số trình duyệt của một vài thiết bị đã hỗ trợ WebGL : stock microB (Nokia N900), WebWorks (BlackBerry PlayBook), firefox for mobile, Sony Ericsson Xperia, Opera Mobile 12 (Android 1.6+), Iphone4S, Ipad2, Ipad3.

- ❖ Kích hoạt chức năng WebGL cho một số trình duyệt :

- Firefox : download và cài đặt phiên bản Firefox 4.0 hoặc phiên bản mới hơn. Vào trình duyệt gõ about:config, sau đó search với webgl, bật cờ kích hoạt WebGL



- Chrome : sử dụng phiên bản Chrome 9 hoặc mới hơn. Vào trình duyệt gõ *about:flags*, tìm dòng webGL và chuyển sang chế độ true.



- Safari : sử dụng phiên bản 4.0 hoặc mới hơn, chạy trên hệ điều hành Mac phiên bản Snow Leopard (OSX 10.6) hoặc mới hơn.

- Download và cài đặt Webkit nightly build : <http://nightly.webkit.org/>
- Vào Terminal gõ :

```
defaults write com.apple.Safari WebKitWebGLEnabled -bool YES
```

- Chạy Freshly-installed Webkit application.

- Safari trên Iphone4S/Ipad2/Ipad3 : đối với thiết bị chạy hệ điều hành iOS 5.1.1 và đã jailbreak, việc kích hoạt chức năng WebGL trên trình duyệt của nó (Safari) là điều rất dễ dàng. Sử dụng phần mềm WebGL Enabler để kích hoạt WebGL trên Safari. Cụ thể từng bước như sau :

- Download WebGL Enabler (file \*.deb):

[http://demoseen.com/webglenabler/com.daeken.webglenabler\\_0.0.2-1\\_iphoneos-arm.deb](http://demoseen.com/webglenabler/com.daeken.webglenabler_0.0.2-1_iphoneos-arm.deb)

- Cài đặt MobileSubstrate trên Cydia
- Cài đặt OpenSSH và restart lại máy.
- Sử dụng WinSCP để chuyển file \*.deb vừa download vào thiết bị. Vào WinSCP nhập địa chỉ IP, username, pass để kết nối desktop với thiết bị di động. Có thể nhập username : root, password : alpin. Sau đó là copy file \*.deb vào thiết bị với đường dẫn thư mục sau :  
“/var/root/media/Cydia/AutoInstall”
- Reboot thiết bị, Cydia tự động cài đặt WebGL Enabler.



Chúng ta có thể thấy biểu tượng WebGL trong mục Settings của thiết bị. Vậy là đã hoàn thành việc kích hoạt WebGL trên **Iphone4S/Ipad2/Ipad3**. Từ đây, chúng ta có thể hiển thị các trang hỗ trợ WebGL mà không cần làm thao tác gì thêm.