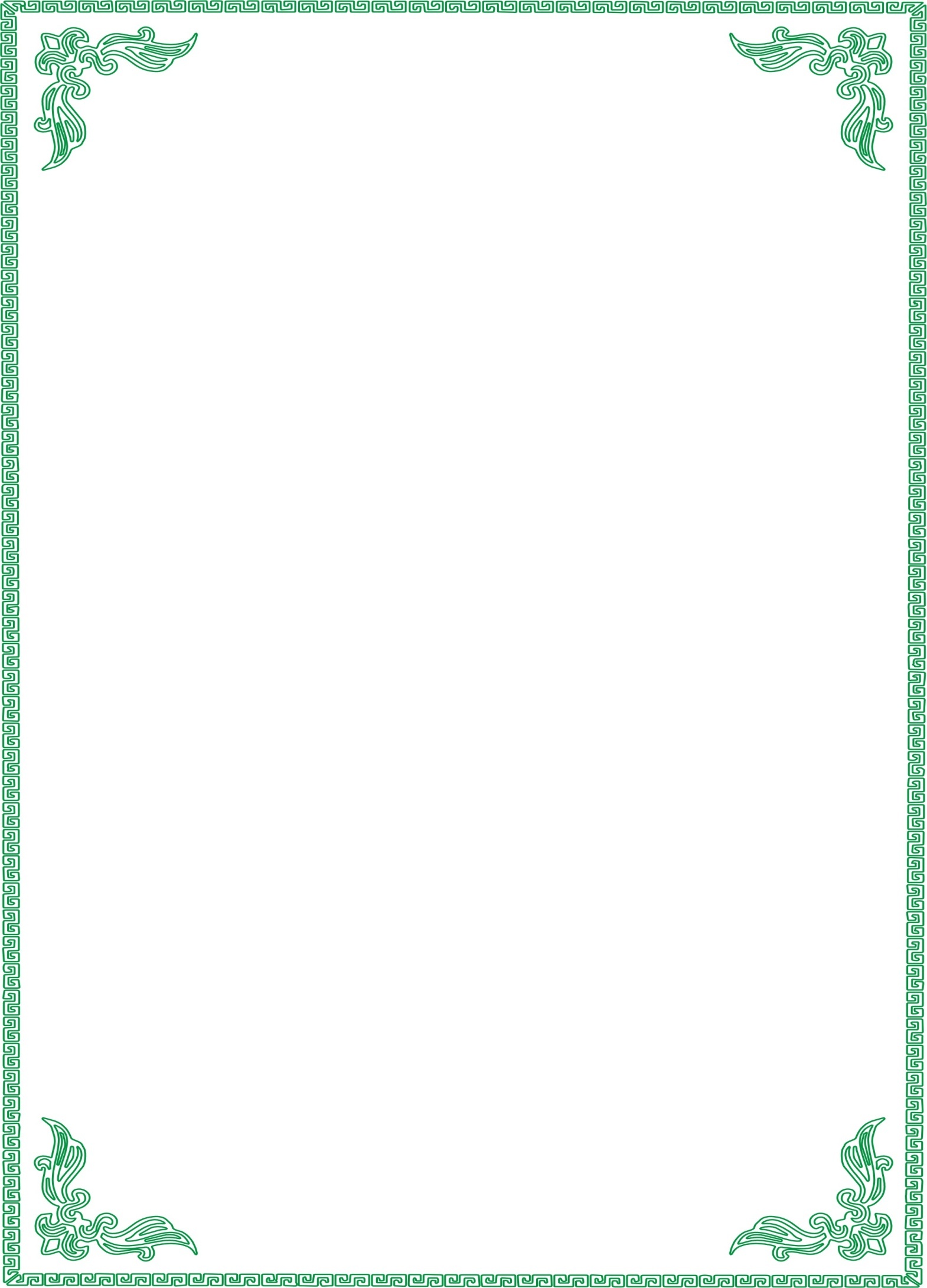
** TRƯỜNG ĐẠI HỌC CÔNG NGHIỆP HÀ NỘI**

**KHOA: CÔNG NGHỆ THÔNG TIN**

- - - - -o0o- - - - -



**BÁO CÁO**

**Môn: CÔNG NGHỆ THỰC TẠI ẢO**

**Đề tài: Mô phỏng tòa nhà A12 ĐHCN HN**

***Giáo viên HD* :** **Vũ Đức Huy**

***Sinh viên thực hiện* : *Nhóm:***

***Hồ Ngọc Trung - 2017607359***

***Văn Phú Thăng - 2017607369***

***Bùi Tuấn Anh - 2017607372***

**Hà Nội – 2018**

**LỜI CẢM ƠN**

Trong quá trình thực hiện đề tài ***“Mô phỏng tòa nhà A5 ĐHCN HN***”, nhóm đã gặp phải không ít khó khăn và trở ngại. Tuy nhiên, với sự giúp đỡ của bạn bè, cùng sự hướng dẫn tận tình, chỉ bảo cặn kẽ của thầy cô, cuối cùng chúng tôi cũng đã hoàn thành cơ bản một số mục tiêu đề ra với tất cả sự cố gắng và nỗ lực. Chúng tôi xin được gửi lời cảm ơn tới những người bạn đã góp ý và giúp đỡ nhóm, và đặc biệt là cảm ơn thầy ***Vũ Đức Huy***, người đã giúp chúng tôi xác định được những yêu cầu và mục tiêu đề tài, truyền dạy các kĩ năng nghiệp vụ cũng như hướng dẫn cho chúng tôi, từ phong cách làm báo cáo đến cách thức thực hiện đề tài. Hi vọng nhóm sẽ tiếp tục nhận được sự đóng góp và đồng hành của các bạn và thầy cô trong những đề tài tiếp theo.

Một lần nữa chúng tôi xin chân thành cảm ơn!

*Sinh viên thực hiện*

***Hồ Ngọc Trung***

***Văn Phú Thăng***

***Bùi Tuấn Anh***

**MỤC LỤC**

[CHƯƠNG 1. TỔNG QUAN VỀ THỰC TẠI ẢO 6](#_Toc407046292)

[1.1  Thực tại ảo là gì ? 6](#_Toc407046293)

[1.2   Lịch sử phát triển của công nghệ thực tại ảo 6](#_Toc407046294)

[1.3   Các thành phần một hệ thống VR 7](#_Toc407046295)

[1.3.1    Phần cứng (Hardware) 7](#_Toc407046296)

[1.3.2    Phần mềm (Software) 8](#_Toc407046297)

[1.4   Các đặc tính chính của VR 8](#_Toc407046298)

[1.5   Một số ứng dụng chính của VR 9](#_Toc407046299)

[1.5.1    Trong y học 9](#_Toc407046300)

[1.5.2    Trong khoa học kỹ thuật 10](#_Toc407046301)

[1.5.3    Trong kiến trúc 11](#_Toc407046302)

[1.5.4    Trong quân sự 11](#_Toc407046303)

[1.5.5    Trong giáo dục 12](#_Toc407046304)

[1.5.6    Trong giải trí 13](#_Toc407046305)

[CHƯƠNG 2. NGÔN NGỮ VRML 14](#_Toc407046306)

[2.1 Giới thiệu về VRML 14](#_Toc407046307)

[2.1.1   VRML là gì ? 14](#_Toc407046308)

[2.1.2   Lịch sử ra đời và phát triển của VRML 14](#_Toc407046309)

[2.1.3   Đặc điểm cơ bản của VRML 15](#_Toc407046310)

[2.2 Các vấn đề cơ bản liên quan đến VRML 16](#_Toc407046311)

[2.2.1 Công cụ soạn thảo và hiển thị VRML 16](#_Toc407046312)

[2.2.2 Tập tin của VRML 17](#_Toc407046313)

[2.2.3 Các nút trong VRML 17](#_Toc407046314)

[2.3   Các kiểu dữ liệu trong VRML 19](#_Toc407046315)

[2.4 Các nút trong VRML 21](#_Toc407046316)

[2.4.1 Nút thông tin (WorldInfo) 23](#_Toc407046317)

[2.4.2 Nút hình dáng (Sharp node) 23](#_Toc407046318)

[2.5 Các đối tượng hình học cơ bản trong VRML 24](#_Toc407046319)

[2.5.1 Hình hộp (Box) 24](#_Toc407046320)

[2.5.2 Hình nón (Cone) 24](#_Toc407046321)

[2.5.3 Hình cầu (Sphere) 25](#_Toc407046322)

[2.5.4 Hình trụ (Cylinder) 25](#_Toc407046323)

[2.6 Các đối tượng hình học phức tạp 26](#_Toc407046324)

[2.6.1 Hiển thị văn bản (Text) 26](#_Toc407046325)

[2.6.2 Xây dựng các đường thẳng trong hệ tọa độ ba chiều 27](#_Toc407046326)

[2.6.3 Xây dựng khung bề mặt trong không gian 29](#_Toc407046327)

[2.6.4 Xây dựng khung lưới và bản đồ trong không gian 31](#_Toc407046328)

[2.7 Các phép biến đổi trong VRML 33](#_Toc407046329)

[2.7.1 Transform 33](#_Toc407046330)

[2.7.2 Phép dịch chuyển (Translation) 34](#_Toc407046331)

[2.7.3 Phép quay (Rotation) 34](#_Toc407046332)

[2.7.4 Phép co giãn (Scale) 35](#_Toc407046333)

[2.7.5 Children 35](#_Toc407046334)

[2.8 Màu sắc và hình ảnh trên đối tượng trong VRML 36](#_Toc407046335)

[2.8.1 Màu sắc trong VRML 36](#_Toc407046336)

[2.8.2 Vẽ hình trên đối tượng 37](#_Toc407046337)

[2.8.3 Dán hình lên đối tượng 38](#_Toc407046338)

[2.8.4 Chiếu phim trên bề mặt đối tượng 39](#_Toc407046339)

[2.9 Âm thanh, ánh sáng, camera, phong cảnh trong VRML 40](#_Toc407046340)

[2.9.1 Âm thanh 40](#_Toc407046341)

[2.9.2 Ánh sáng 42](#_Toc407046342)

[2.9.3   Camera 46](#_Toc407046343)

[2.9.4   Phong cảnh nền và môi trường 47](#_Toc407046344)

[2.10 Nhóm các đối tượng trong VRML 50](#_Toc407046345)

[2.10.1 Anchor 50](#_Toc407046346)

[2.10.2 Group 51](#_Toc407046347)

[2.10.3 Switch 51](#_Toc407046348)

[2.10.4 Transform 52](#_Toc407046349)

[2.10.5 Billboard 52](#_Toc407046350)

[2.10.6 Inline 52](#_Toc407046351)

[2.11 Tái sử dụng các đối tượng trong VRML 53](#_Toc407046352)

[2.11.1 Inline 53](#_Toc407046353)

[2.11.2 DEF 53](#_Toc407046354)

[2.11.3 PROTO 54](#_Toc407046355)

[CHƯƠNG 3: ỨNG DỤNG VRML - XÂY DỰNG CHƯƠNG TRÌNH MÔ PHỎNG KHU PHỐ GIAO THÔNG ĐÔ THỊ 56](#_Toc407046356)

[3.1 Yêu cầu bài toán 56](#_Toc407046357)

[3.2 Khảo sát và phân tích 56](#_Toc407046358)

[3.3 Xây dựng chương trình mô phỏng 57](#_Toc407046359)

[3.3.1 Các phương tiện 57](#_Toc407046360)

[3.3.2 Người 60](#_Toc407046361)

[3.3.3 Đường 60](#_Toc407046362)

[3.3.4 Dải phân cách 61](#_Toc407046363)

[3.3.5 Vỉa hè 62](#_Toc407046364)

[3.3.6 Toàn cảnh 65](#_Toc407046365)

[3.4 Kết quả 67](#_Toc407046366)

[KẾT LUẬN 68](#_Toc407046367)

[TÀI LIỆU THAM KHẢO 69](#_Toc407046368)

**CHƯƠNG 1. TỔNG QUAN VỀ THỰC TẠI ẢO**

**1.1          Thực tại ảo là gì ?**

Thực tại ảo (VR- Virtual Reality) là công nghệ sử dụng các kỹ thuật mô hình hóa không gian, cùng với sự hỗ trợ của các thiết bị đa phương tiện để xây dựng một thế giới mô phỏng (môi trường ảo) bằng máy tính nhằm đưa người sử dụng vào một thế giới nhân tạo với không gian như thật. Người sử dụng sẽ không như người quan sát bên ngoài, mà trở thành một phần của hệ thống. Thế giới “nhân tạo” này không tĩnh tại, mà lại phản ứng, thay đổi theo ý muốn của người sử dụng nhờ hành động, lời nói,... Người sử dụng có thể nhìn thấy sự vật thay đổi trên màn hình theo ý muốn của họ và cảm nhận bằng các giác quan bởi sự mô phỏng này.

Trong thực tế, người dùng không những nhìn thấy đối tượng đồ họa 3D nổi, điều khiển được đối tượng như quay, di chuyển,… trên màn hình, mà còn sờ và cảm thấy chúng như có thật. Ngoài khả năng nhìn (thị giác), nghe (thính giác), sờ (xúc giác), thực tại ảo còn có khả năng tạo các cảm giác khác như ngửi (khứu giác), nếm (vị giác), tuy nhiên hiện nay các cảm giác này ít được sử dụng đến.

Hiện nay có khá nhiều khái niệm về thực tại ảo, một trong các định nghĩa được chấp nhận rộng rãi là của C. Burdea và P. Coiffet: “Thực tại ảo là một hệ thống giao diện cấp cao giữa người sử dụng và máy tính. Hệ thống này mô phỏng các sự vật và hiện tượng theo thời gian thực và tương tác với người sử dụng qua tổng hợp các kênh cảm giác. Đó là ngũ giác gồm: thị giác, thính giác, xúc giác, khứu giác, vị giác”.

**1.2         Lịch sử phát triển của công nghệ thực tại ảo**

Khái niệm thực tại ảo đã có trong nhiều thập kỷ trước đây, nhưng nó thực sự được nhận thức trong những năm 1990.

Vào năm 1960, nhà quay phim Morton Heilig (Mỹ) đã phát minh ra thiết bị mô phỏng SENSORAMA bao gồm một màn hình lập thể, quạt, máy tạo mùi, loa và ghế chuyển động.

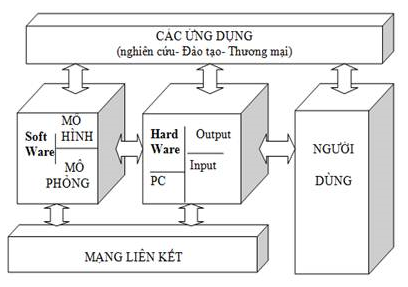
Vào năm 1961, những kỹ sư của Công ty Philco là những người phát triển thiết bị HMD đầu tiên được gọi là Headsight. Thiết bị này được sử dụng trong các tình huống nguy hiểm, quan sát một môi trường thực tế từ xa, điều chỉnh góc quay camera bằng cách quay đầu. Phòng thí nghiệm Bell đã sử dụng HMD tương tự cho những phi công lái máy bay trực thăng. Họ liên kết HMD với những camera hồng ngoại gắn bên ngoài máy bay giúp phi công có thể nhìn rõ ngay cả trong môi trường thiếu ánh sáng.

Vào năm 1965, nhà khoa học máy tính Ivan Sutherland đã đưa ra một hệ thống mà ông ta gọi là “Ultimate Display”. Với hệ thống hiển thị này, một người có thể thấy một thế giới ảo hiện ra như thế giới vật lý thật. Điều này đã định hướng toàn bộ tầm nhìn về VR. Hệ thống của Suntherland bao gồm:

* Một HMD dùng để quan sát thế giới ảo.
* Một máy tính để duy trì các mô hình trong thời gian thực
* Hệ thống hỗ trợ các khả năng cho người sử dụng để thao tác những đối tượng thực tế một cách trực quan nhất.

VR chỉ thực sự được phát triển ứng dụng rộng rãi trong những năm gần đây nhờ vào sự phát triển của tin học (phần mềm) và máy tính (phần cứng).

**1.3         Các thành phần một hệ thống VR**

Tổng quát một hệ thống VR bao gồm những thành phần sau:

*Hình 1.1 Các thành phần của một hệ thống VR*

**1.3.1    Phần cứng (Hardware)**

Phần cứng của một VR bao gồm:

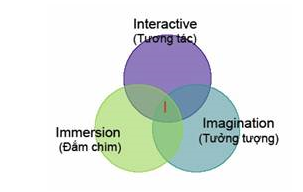
* Máy tính (PC hay Workstation với cấu hình đồ họa mạnh).
* Các thiết bị đầu vào (Input devices): Bộ dò vị trí (position tracking) để xác định vị trí quan sát. Bộ giao diện định vị (Navigation interfaces) để di chuyển vị trí người sử dụng. Bộ giao diện cử chỉ (Gesture interfaces) như găng tay dữ liệu (data glove) để người sử dụng có thể điều khiển đối tượng.
* Các thiết bị đầu ra (Output devices): gồm hiển thị đồ họa (như màn hình, HDM,..) để nhìn được đối tượng 3D nổi. Thiết bị âm thanh (loa) để nghe được âm thanh vòm (như Hi-Fi, Surround,...). Bộ phản hồi cảm giác (Haptic feedback như găng tay,...) để tạo xúc giác khi sờ, nắm đối tượng. Bộ phản hồi xung lực (Force Feedback) để tạo lực tác động như khi đạp xe, đi đường xóc,...

**1.3.2    Phần mềm (Software)**

Phần mềm luôn là linh hồn của VR cũng như đối với bất cứ một hệ thống máy tính hiện đại nào. Về mặt nguyên tắc có thể dùng bất cứ ngôn ngữ lập trình hay phần mềm đồ họa nào để mô hình hóa (modelling) và mô phỏng (simulation) các đối tượng của VR. Ví dụ như các ngôn ngữ (miễn phí) OpenGL, C++, Java3D, VRML, X3D,... hay các phần mềm thương mại như WorldToolKit, PeopleShop,... Phần mềm của bất kỳ VR nào cũng phải bảo đảm 2 công dụng chính: Tạo hình vào Mô phỏng. Các đối tượng của VR được mô hình hóa nhờ chính phần mềm này hay chuyển sang từ các mô hình 3D (thiết kế nhờ các phần mềm CAD khác như AutoCAD, 3D Studio,...). Ngoài ra, phần mềm VR cần có khả năng mô phỏng động học, động lực học, và mô phỏng phản ứng của đối tượng.

**1.4        Các đặc tính chính của VR**

Đặc tính chính của VR là Tương tác (Interactive) và Đắm chìm (Immersion). Tuy nhiên, VR không chỉ là một hệ thống tương tác Người - Máy, mà các ứng dụng của nó còn liên quan tới việc giải quyết các vấn đề thật trong kỹ thuật, y học, quân sự,... Các ứng dụng này phụ thuộc rất nhiều vào khả năng tưởng tượng (Imagination) của con người. Do đó có thể coi VR là tổng hợp của 3 yếu tố: Tương tác - Đắm chìm - Tưởng tượng (“3I” trong tiếng Anh: Interactive – Immersion - Imagination).



*Hình 1.2 Các đặc tính chính của VR*

**1.5         Một số ứng dụng chính của VR**

Tại các nước phát triển, VR được ứng dụng trong mọi lĩnh vực: Khoa học kỹ thuật, kiến trúc, quân sự, giải trí,... và đáp ứng mọi nhu cầu: Nghiên cứu - Giáo dục - Thương mại. Y học là lĩnh vực ứng dụng truyền thống của VR. Ngoài ra, VR cũng đã được ứng dụng trong giáo dục, nghệ thuật, giải trí. Đặc biệt trong lĩnh vực quân sự, VR đã được ứng dụng rất nhiều ở các nước phát triển hiện nay. Bên cạnh các ứng dụng truyền thống ở trên, cũng có một số ứng dụng mới nổi lên trong thời gian gần đây của VR như: ứng dụng trong sản xuất, ứng dụng trong ngành robot, ứng dụng trong hiển thị thông tin (thăm dò dầu mỏ, hiển thị thông tin khối, ...). VR có tiềm năng ứng dụng vô cùng lớn, hầu hết các lĩnh vực “có thật” trong cuộc sống đều có thể ứng dụng “thực tại ảo” để nghiên cứu và phát triển hoàn thiện hơn.

**1.5.1    Trong y học**

Thực tại ảo giải quyết được rất nhiều vấn đề trong y học: cung cấp môi trường thực hành cho nghiên cứu và học tập, rất hữu ích trong việc mô phỏng các ca phẫu thuật nhằm giảm tối đa rủi ro trong thực tế.



*Hình 1.3 Một ca phẫu thuật trong thực tại ảo*

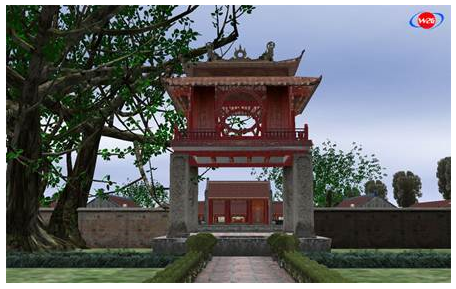
**1.5.2    Trong khoa học kỹ thuật**

Với sự trợ giúp của thực tại ảo, ngày nay, con người không những có thể xem được hình ảnh trực quan của các thiết bị cần sản xuất mà thậm chí người ta còn có khả năng sử dụng hay thay đổi các chi tiết của các thiết bị đó. Việc này nhằm giúp cho các nhà khoa học và các kỹ sư thuận lợi hơn trong việc tạo ra một sản phẩm hợp ý muốn mà không cần tốn nhiều chi phí.

*Hình 1.4 Các kỹ sư đang thay đổi các chi tiết cho chiếc xe hơi ảo*

**1.5.3    Trong kiến trúc**

Trong nhiều năm trở lại đây, thực tại ảo đã được sử dụng để xây dựng mô hình của các dự án kiến trúc trước khi các dự án này đưa vào thực tế, nhằm giúp cho người sử dụng có cái nhìn tổng quan và chi tiết về các dự án đó. Bên cạnh đó, thực tại ảo cũng được sử dụng để tái hiện lại các công trình kiến trúc cổ, nhằm lưu giữ lại các di sản văn hóa...



*Hình 1.5 Một góc của Văn Miếu Quốc Tử Giám trên mô hình 3D*

**1.5.4    Trong quân sự**

Với việc phát triển của VR, các binh sĩ sẽ được huấn luyện một cách trực quan nhất các kĩ năng cần thiết như: lái máy bay, lái xe tăng,... trước khi tham gia công việc thực tế. Điều này vừa bảo đảm an toàn cho binh sĩ, vừa tiết kiệm được chi phí cho các khóa huấn luyện thực tế. Quân đội Mỹ đã phát triển **một game đặc biệt nhằm huấn luyện binh sĩ chống lại khủng bố dưới dạng chiến thuật thực tại ảo (Hình 1.6).** Đây sẽ là một game rất sống động, có tính hành động cao với môi trường và bối cảnh bám sát với thực tế. Những người lính sẽ phải vận dụng tất cả những kỹ năng đã được rèn giũa trong quân đội.



*Hình 1.6 Ứng dụng của VR trong quân sự*

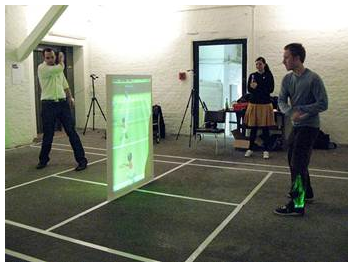
**1.5.5    Trong giáo dục**

Ở các nước phương Tây việc ở nhà học qua Internet không còn là điều mới mẻ. Công nghệ VR sẽ làm cho việc này trở nên thú vị hơn rất nhiều. Người học có thể điều khiển một nhân vật đại diện cho mình đi lại trong một trường học ảo được xây dựng trên máy tính. Người học cũng có thể tham gia vào bất cứ lớp học ảo nào mà họ thích và nói chuyện với những thành viên khác trong lớp.



*Hình 1.7 Cảnh sinh hoạt trong lớp học ảo*

**1.5.6    Trong giải trí**

Game thực tại ảo hiện nay đã trở thành một ngành công nghiệp thu được nhiều lợi nhuận. Ở nước ta hiện nay thì game thực tại ảo chưa được biết tới nhiều song ở một số nước phát triển thì đây là một ngành giải trí thu lợi nhuận khổng lồ, ví dụ các nước Mỹ, Nhật, Anh, ...

*Hình 1.8 Game Nintendo Wii*

Như vậy thực tại ảo có ứng dụng trong hầu hết các lĩnh vực của cuộc sống. Qua đó cũng nhận thấy được ý nghĩa to lớn của việc ứng dụng thực tại ảo. Với những vấn đề khó khăn, nếu không có thực tại ảo thì rất khó giải quyết hoặc hiệu quả không cao mà chi phí tốn kém.

**CHƯƠNG 2. NGÔN NGỮ VRML**

**2.1 Giới thiệu về VRML**

**2.1.1        VRML là gì ?**

VRML (Virtual Reality Modeling Language) là ngôn ngữ mô hình hóa thực tại ảo, một định dạng tập tin được sử dụng trong việc mô tả thế giới thực và các đối tượng đồ họa tương tác ba chiều, sử dụng mô hình phân cấp trong việc thể hiện tương tác với các đối tượng của mô hình, được thiết kế dùng trong môi trường Internet, Intranet và các hệ thống máy khách cục bộ (local client) mà không phụ thuộc vào hệ điều hành.

Các ứng dụng 3D của VRML có thể truyền đi một cách dễ dàng trên mạng với kích thước khá nhỏ so với băng thông, phần lớn giới hạn trong khoảng 100 - 200KB. Nếu HTML là định dạng văn bản thì VRML là định dạng đối tượng 3D có thể tương tác và điều khiển thế giới ảo.

Hiện nay, VRML có lợi thế là sự đơn giản, hỗ trợ dịch vụ Web3D, có cấu trúc chặt chẽ, với khả năng mạnh mẽ, giúp cho việc xây dựng các ứng dụng đồ họa ba chiều một cách nhanh chóng và chân thực nhất.

VRML là một trong những chuẩn trao đổi đa năng cho đồ họa ba chiều tích hợp và truyền thông đa phương tiện, được sử dụng trong rất nhiều lĩnh vực ứng dụng, chẳng hạn như trực quan hóa các khái niệm khoa học và kỹ thuật, trình diễn đa phương tiện, giải trí và giáo dục, hỗ trợ web và chia sẻ các thế giới ảo. Với mục đích xây dựng định dạng chuẩn cho phép mô tả thế giới thực trên máy tính và cho phép chạy trên môi trường web, VRML đã trở thành chuẩn ISO từ năm 1997.

**2.1.2       Lịch sử ra đời và phát triển của VRML**

Năm 1994, lần đầu tiên VRML được thảo luận tại hội nghị WWW, Gieneva, Thụy Sĩ. Tim Berners-Lee và Dave Raggett đã tổ chức ra phiên họp có tên là Birds of a Feather (BOF) để mô tả giao diện thực tại ảo trên WWW. Nhiều thành viên tham dự, phiên họp BOF đã mô tả nhiều dự án thực hiện việc xây dựng các công cụ hiển thị đồ họa 3D cho phép có nhiều thao tác hữu ích trên Web. Những thành viên này đã nhất trí đồng ý sự cần thiết cho các công cụ này có một ngôn ngữ chung, phổ biến cho định dạng, xác định việc mô tả thế giới 3D và các siêu liên kết WWW. Vì thế, cụm từ “the Virtual Reality Markup Language” ra đời, từ “Markup” sau đó đã được đổi thành “Modelling” để phản ánh bản chất tự nhiên của VRML.

Sau phiên họp BOF một thời gian ngắn thì tổ chức WWW-VRML được thành lập để tập trung vào xây dựng phiên bản VRML đầu tiên.

Vào tháng 3/ 1995, Công ty Silicon Graphics cộng tác với hãng Sony Research và Mitra để đưa ra phiên bản mới cho VRML. Bản đệ trình của Silicon Graphics có tên là “Moving Worlds” gửi đến tổ chức Request for Proposals cho việc xây dựng phiên bản mới VRML, bản đệ trình này là một minh chứng cho sự cộng tác thành công của tất cả các thành viên của Silicon Graphics, Sony và Mitra. Năm 1996 tại New Orleans, phiên bản đầu tiên của VRML 2.0 được đưa ra.

Vào tháng 7/1996, tổ chức tiêu chuẩn quốc tế (ISO) đã thống nhất ý kiến lấy phiên bản năm 1996 của VRML 2.0 để đưa ra xem xét vào tháng 4/1997. Sau khi bỏ phiếu về chuẩn ISO thì VRML97 được đưa ra như một chuẩn ISO vào năm 1997.

**2.1.3       Đặc điểm cơ bản của VRML**

Tiêu chuẩn cho việc xác định đối tượng 3D, quang cảnh và cho sự liên kết các mô hình với nhau là:

* Không phụ thuộc phần cứng: có thể chạy trên các máy tính do các nhà sản xuất khác nhau chế tạo.
* Có thể mở rộng: có thể chấp nhận các lệnh mới do người sử dụng thêm vào hoặc quy định.
* Thao tác được thế giới ảo thông qua môi trường Internet có băng thông thấp.

VRML được thiết kế dành riêng cho việc hiển thị thế giới 3D và không phải là sự mở rộng của HTML.

## 2.2 Các vấn đề cơ bản liên quan đến VRML

* + 1. **Công cụ soạn thảo và hiển thị VRML**

Bộ soạn thảo VRML cho phép người dùng gõ mã VRML. Có thể sử dụng một trình soạn thảo văn bản bất kỳ như notepad, Word,... Tuy nhiên, VRML Pad là phần mềm thông dụng giúp soạn thảo và cho xem trực tiếp kết quả mà không cần qua trình duyệt Internet.

Trình duyệt VRML cũng giống như trình duyệt Internet (Internet Explorer hay Fire Fox) và được tích hợp trong các trình duyệt này. Các file chỉ có thể đọc được nếu hệ thống có trình duyệt VRML.

Để hiển thị các file VRML, có thể sử dụng trình duyệt Cortona 3D Viewer của hãng Parallel Graphics. Phần mềm này sẽ giúp người dùng thuận tiện hơn khi xem các mô hình ảo trên máy tính một cách trực quan sinh động.

Yêu cầu trước khi cài đặt Cortona 3D Viewer:

* Hệ điều hành Microsoft Windows XP / Vista / 7.
* Trình duyệt Web Internet Explorer 6.0 trở lên, Google Chrome 9.0 trở lên, Netscape Navigator 8.0 trở lên, Mozilla Firefox 1.5 trở lên, Opera 8.5 trở lên.
* CPU Pentium® II 300 MHz trở lên.
* RAM tối thiểu 64 MB.
* Độ phân giải màn hình tối thiểu 1024x768.
* Card đồ họa hỗ trợ 3D và cài đặt DirectX 9.

Cortona 3D Viewer tương thích với hầu hết các trình duyệt như Internet Explorer, Netscape Browser, Mozilla, Mozilla Firefox và các công cụ văn phòng như Word, PowerPoint...

Tính năng của Cortona 3D Viewer là trình diễn toàn bộ mô hình 3D trên máy tính một cách hoàn hảo với các hiệu ứng trên nhiều hệ thống như Flash, DirectX9, MPEG4... Khi truy xuất vào một ứng dụng VRML, toàn bộ hình mô phỏng sẽ được trình diễn tương tác trên nền 3D dạng mở. Rất ấn tượng và bắt mắt.

* + 1. **Tập tin của VRML**

Tập tin của VRML có phần mở rộng là “.wrl” với các phần như sau:

* **Header**: dùng để nhận dạng tập tin VRML và cách mã hóa. Header của file VRML bắt đầu bằng dấu #. Ngoài lần xuất hiện đầu tiên ra thì dấu # đánh dấu những gì theo sau nó là phần chú thích. File tiêu đề của VRML có dạng: *#VRML V1.0 ascii* dành riêng cho phiên bản VRML 1.0 và *#VRML V2.0 utf-8* dành cho phiên bản 2.0.
* **Scene Graph**: chứa những node mô tả các đối tượng và các thuộc tính đi kèm. Nó gần như một cây phả hệ gồm các nhóm đối tượng.
* **Prototype**: cho phép một tập các nút kiểu VRML được mở rộng bởi người sử dụng. Các định danh kiểu này có thể được bao hàm trong file (mà chúng được sử dụng) hay định nghĩa ở bên ngoài (file đó).
* **Event routing**: một số nút có thể phát sinh những sự kiện đáp trả những thay đổi môi trường do tương tác phía người dùng. “Event routing” cho phép một sự kiện phát sinh được truyền đến các “đích”- những nút trong hệ thống, từ đó gây ra những thay đổi cho riêng nút đó và hệ thống.
  + 1. **Các nút trong VRML**

Tập tin VRML được xây dựng dựa trên tập các đối tượng nhằm đến các mục đích khác nhau. Thông thường các đối tượng có các thuộc tính vật lý của mình như hình dạng, màu sắc, tọa độ điểm, ... Để mô tả cho các đối tượng của thế giới thật, VRML sử dụng thuật ngữ “Nút - Node” để biểu diễn chúng.

Nút là khối cơ sở của tập tin VRML dùng để mô tả những đối tượng mà thuộc tính của chúng được định nghĩa trong nút đó. Nút có thể là các đối tượng hình học như hình hộp, hình nón, hình trụ … hay các đối tượng khác như màu sắc, ánh sáng, âm thanh. Sự tồn tại của nút trong tập tin VRML có thể là một cấu trúc cơ bản đứng đơn lẻ hoặc có thể chứa nhiều các nút có liên hệ với nhau.

Dữ liệu của nút được lưu giữ bởi các trường (Field) trong nút, tuy nhiên ta có thể khai báo chỉ một nút trong file nhưng không thể chỉ đưa ra một trường đơn lẻ mà bắt buộc phải để trong một nút nào đó. Về một khía cạnh nào đó nút tương đương với một lớp (class) trong các ngôn ngữ lập trình hướng đối tượng (Java). VRML bao gồm 54 nút khác nhau và được phân loại làm 9 nhóm chính dựa trên chức năng và các hàm của các nút. Bao gồm:

* **Grouping Nodes**: Nhóm các nút nhóm.
  + Anchor
  + Billboard
  + Collision
  + Group
  + Transform.
* **Special Groups Nodes**: Nhóm các nút nhóm đặc biệt.
  + Inline
  + LOD
  + Switch.
* **Sensors Nodes**: Nhóm các nút cảm biến.
  + CylinderSensor
  + PlaneSensor
  + ProximitySensor
  + SphereSensor
  + TimeSensor
  + TouchSensor
  + VisibilitySensor.
* **Geometry Nodes**: Nhóm các nút đối tượng hình học.
  + Box
  + Cone
  + Cylinder
  + ElevationGrid
  + Extrusion
  + IndexedFaceSet
  + IndexedLineSet
  + PointSet
  + Sphere
  + Text.
* **Geometry Properties Nodes**: Nhóm các nút thuộc tính hình học.
  + Color
  + Coordinate
  + Normal
  + TextureCoordinate.
* **Appearance Nodes**: Nhóm các nút mô tả hiển thị.
  + Appearance
  + FontStyle
  + ImageTexture
  + Material
  + MovieTexture
  + PixelTexture
  + TextureTransform.
* **Interpolators Nodes**: Nhóm các nút nội suy.
  + ColorInterpolator
  + CoordinateInterpolator
  + NormalInterpolator
  + OrientationInterpolator
  + PositionInterpolator
  + ScalarInterpolator.
* **Bindable Nodes**: Nhóm các nút có thể ghép được.
  + Background
  + Fog
  + NavigationInfo
  + Viewpoint.

Tên các nút trong VRML thường bắt đầu bằng chữ in hoa và chỉ có thể là một trong các tên chuẩn do VRML cung cấp, các trường của nút thường bắt đầu là chữ thường, mỗi loại nút có các trường khác nhau. Giá trị của trường có thể là các giá trị thực hoặc các bộ giá trị thực hoặc có thể là một nút cơ bản. Có thể hình dung các nút như các lớp trong lập trình hướng đối tượng. VRML không cho phép định nghĩa thêm các nút mới mà chỉ được dùng các nút cơ bản trong chín nhóm nút đã nêu.

**2.3        Các kiểu dữ liệu trong VRML**

Trong VRML, các nút có thể chứa các nút khác và có thể chứa các trường. Mỗi trường có các kiểu dữ liệu khác nhau. Sau đây là mô tả của chúng:

* **SFBool**

Đây là giá trị kiểu logic, có thể nhận giá trị “TRUE” hay “FALSE”.

* **SFColor & MFColor**

SFColor là trường chứa màu sắc, được tạo nên từ ba số thực từ 0 đến 1 tương ứng với các giá trị màu red, green và blue (RGB) (ví dụ: 0 1 0 là green). MFColor là một tập hợp gồm các màu sắc (ví dụ: [0 1 0, 1 0 0, 0 0 1] là tập hợp gồm ba màu green, red và blue).

* **SFFloat & MFFloat**

SFFloat là giá trị kiểu số thực (ví dụ: 7.5). MFFloat là một tập các giá trị số thực (ví dụ: [1.0, 3.4, 76.54]).

* **SFImage**

SFImage là hình ảnh hai chiều được tạo nên từ các điểm ảnh. Nó bao gồm:

* Hai số nguyên, đại diện chiều rộng và chiều cao của hình ảnh cũng chính là số điểm nằm trên chiều rộng và chiều cao của hình.
* Một số nguyên biểu thị số thành phần của hình ảnh. Số thành phần là 1 tức hình ảnh trắng đen, là 2 cũng tương tự nhưng có thêm độ trong suốt, là 3 thì hình sẽ có màu (RGB), và 4 là hình ảnh có màu kết hợp với độ trong suốt.
* Sau đó, là các con số được biểu diễn dưới dạng thập lục phân đại diện cho màu sắc của từng điểm ảnh. Ví dụ 0xFF là màu trắng trong hình ảnh một thành phần và 0xFF00007F sẽ là màu đỏ nhạt trong hình ảnh bốn thành phần.

Các điểm được chỉ rõ theo thứ tự từ trái sang phải, từ trên xuống dưới.

Ví dụ: 1 2 1 0xFF 0x00 *# Hình ảnh một thành phần gồm 2 điểm màu trắng và đen.*

* **SFInt32 & MFInt32**

Giá trị kiểu số nguyên hoặc dãy các số nguyên 32 bit có thể theo dạng thập phân hoặc thập lục phân. Các số thập lục phân 32 bit bắt đầu bằng “0x”, ví dụ 0xFF là 255 hệ thập phân.

* **SFNode & MFNode**

SFNode là một nút đơn lẻ và MFNode là danh sách các nút. Trường **children** với nhiều nút là kiểu MFNode.

* **SFRotation & MFRotation**

Các trường này chỉ ra một phép quay xung quanh một trục, nó được tạo nên từ 4 số thực, ba số thực đầu tiên chỉ rõ các toạ độ X, Y và Z cho các vectơ tương ứng với trục để quay, số thứ tư là góc (tính bằng radian) để quay. SFRotation là một tập, MFRotation là một danh sách các tập.

* **SFString & MFString**

Loại này chứa danh sách các kí tự trong tập kí tự **utf-8**, Nói đơn giản thì SFString là một xâu kí tự và MFString là tập các xâu kí tự. Ví dụ SFString là “Hello” (các kí tự nằm trong dấu nháy kép) còn MFString thì tham số truyền vào có dạng là [“Hello” “VRML”].

* **SFTime & MFTime**

Kiểu dữ liệu đặc trưng cho một mốc thời gian hoặc danh sách các mốc thời gian.

* **SFVec2f & MFVec2f**

Các vectơ 2D đơn lẻ hoặc danh sách các vectơ 2D. Một vectơ 2D được tạo bởi một cặp các số thực.

* **SFVec3f & MFVec3f**

Một vectơ 3D hoặc một danh sách các vectơ 3D. Một vectơ 3D là một bộ 3 số thực.

## 2.4 Các nút trong VRML

VRML bao gồm 54 nút khác nhau và được phân loại làm 9 nhóm chính dựa trên chức năng và các hàm của các nút. Bao gồm:

* **Grouping Nodes**: Nhóm các nút nhóm.
* Anchor
* Billboard
* Collision
* Group
* Transform.
* **Special Groups Nodes**: Nhóm các nút nhóm đặc biệt.
* Inline
* LOD
* Switch.
* **Sensors Nodes**: Nhóm các nút cảm biến.
* CylinderSensor
* PlaneSensor
* ProximitySensor
* SphereSensor
* TimeSensor
* TouchSensor
* VisibilitySensor.
* **Geometry Nodes**: Nhóm các nút đối tượng hình học.
* Box
* Cone
* Cylinder
* ElevationGrid
* Extrusion
* IndexedFaceSet
* IndexedLineSet
* PointSet
* Sphere
* Text.
* **Geometry Properties Nodes**: Nhóm các nút thuộc tính hình học.
* Color
* Coordinate
* Normal
* TextureCoordinate.
* **Appearance Nodes**: Nhóm các nút mô tả hiển thị.
* Appearance
* FontStyle
* ImageTexture
* Material
* MovieTexture
* PixelTexture
* TextureTransform.
* **Interpolators Nodes**: Nhóm các nút nội suy.
* ColorInterpolator
* CoordinateInterpolator
* NormalInterpolator
* OrientationInterpolator
* PositionInterpolator
* ScalarInterpolator.
* **Bindable Nodes**: Nhóm các nút có thể ghép được.
* Background
* Fog
* NavigationInfo
* Viewpoint.

### 2.4.1 Nút thông tin (WorldInfo)

Nút **WorldInfo** là nút chứa thông tin chung về thế giới ảo như tiêu đề của thế giới hay một chuỗi thông tin về tác giả hoặc về nội dung của tập tin, nút này không ảnh hưởng đến hình ảnh cũng như các sự kiện trong thế giới (nút này có thể không được sử dụng trong tập tin VRML). Các trường của nút này được thiết kế để lưu trữ tên hoặc tiêu đề của thế giới để trình duyệt có thể hiển thị cho người dùng hoặc phục vụ cho các công cụ tìm kiếm. Sau đây là một ví dụ của nút **WorldInfo**:

WorldInfo {

            title “Hello VRML”

            info [“Virtual Reality Modeling Language”

 “Nguyen Thanh Minh”]

}

**2.4.2 Nút hình dáng (Sharp node)**

Nút **Sharp** là nút cơ sở dùng để chứa các đối tượng hình học thông qua “Geometry nodes” và các thuộc tính cần thể hiện của đối tượng hình học đó qua “Appearance nodes”. Điều đó có nghĩa nếu ta muốn tạo ra bất kỳ một đối tượng nào thì nút hình dáng của đối tượng đó phải được tạo ra trước.

Ví dụ:

**Shape** **{**

**appearance** **Appearance** **{**

**material** **Material** {}

**}**

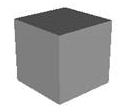
**geometry** **NULL**

}

**2.5 Các đối tượng hình học cơ bản trong VRML**

Các đối tượng hình học trong thế giới ảo thường được cấu tạo từ các đối tượng hình học cơ bản như hình hộp, hình tròn, hình trụ, hình cầu … Và VRML cung cấp sẵn cho chúng ta một số các đối tượng hình học đơn giản đó dưới dạng các nút như **Box** (hình hộp), **Cone** (hình nón), **Cylinder** (hình trụ tròn), **Sphere** (hình cầu)…

* + 1. **Hình hộp (Box)**

Ví dụ:

**Shape** {

geometry **Box**{ size 2.0 2.0 2.0 }

}

Tham số:

* **size** X Y Z: kích thước hình hộp (chiều rộng, chiều cao, chiều sâu).

### 2.5.2 Hình nón (Cone)

Ví dụ:

Shape {

geometry Cone {

height 2.0

bottomRadius 1.0

bottom TRUE

side TRUE

}

}

Các tham số:

* **height** X: chiều cao của hình nón.
* **bottomRadius** Y: bán kính của đáy.
* **bottom** TRUE / FALSE: hiện / ẩn đáy.
* **side** TRUE / FALSE: hiện / ẩn mặt bên.

### 2.5.3 Hình cầu (Sphere)

Ví dụ:

Shape {

geometry Sphere {

radius 1.0

}

}

Tham số:

* **Radius** X: bán kính mặt cầu.

### 2.5.4 Hình trụ (Cylinder)

Ví dụ:

Shape {

geometry Cylinder{

height 2.0

radius 1.0

bottom TRUE

top TRUE

side TRUE

}

}

Các tham số:

* **height** X: chiều cao.
* **radius** Y: bán kính mặt đáy.
* **bottom** TRUE / FALSE: hiện / ẩn mặt đáy dưới.
* **top** TRUE / FALSE: hiện / ẩn mặt đáy trên.
* **side** TRUE / FALSE: hiện / ẩn các mặt bên.
  1. **Các đối tượng hình học phức tạp**

### 2.6.1 Hiển thị văn bản (Text)

Nút này được thiết kế nhằm cho phép hiển thị một đoạn văn bản 2D trong thế giới ảo đang xây dựng. Nút này tương đối đơn giản nhưng nếu chúng ta thao tác với thuộc tính **fontStyle** (là nút **FontStyle** trong VRML)của nó thì lại khá phức tạp.

Ví dụ:

Shape {

geometry Text{

string [“ITqnu” “ntm”]

fontStyle FontStyle{

            size 1

            family “ARIAL”

            style “BOLD”

            spacing 1

             justify “MIDDLE”

             horizontal TRUE

             leftToRight TRUE

             topToBottom TRUE

             }

length [1 1]

maxExtent 5.0

}

}

Các tham số:

* **string** [“Nội dung văn bản”]: hiển thị các xâu trong cặp dấu “[]”.
* **length** [X1 X2]: độ dài của các xâu trong “string”.
* **maxExtent** Y: giới hạn kéo dãn của các dòng.

Các tham số của nút **FontStyle** (nếu chúng ta muốn đơn giản nút **Text** thì có thể đặt thuộc tính **fontStyle** này bằng NULL):

* **family**: tên font sử dụng.
* **style** BOLD / ITALIC / BOLDITALIC / PLAIN: kiểu chữ.
* **size** M: chiều cao, kích thước của chữ.
* **spacing** N: khoảng cách giữa các dòng.
* **justify** BEGIN / MIDDLE / END: canh lề trái / giữa / phải cho các dòng.
* **horizontal** TRUE / FALSE: trình bày ngang / dọc màn hình.
* **leftToRight** TRUE / FALSE: trình bày từ trái sang phải (TRUE) hoặc từ phải sang trái (FALSE).
* **topToBottom** TRUE / FALSE: trình bày từ trên xuống dưới (TRUE) hoặc từ dưới lên trên (FALSE).

### 2.6.2 Xây dựng các đường thẳng trong hệ tọa độ ba chiều

Nút **IndexedLineSet** là nút vẽ một tập hợp các đường thẳng trong hệ tọa độ không gian ba chiều của thế giới VRML với tập hợp các màu tương ứng cho các đường thẳng đó.

Cú pháp:

IndexedLineSet{

            coord Coordinate{}

            coordIndex

            color Color{}

            colorIndex[]

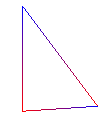
            colorPerVerter TRUE

}

Nút này có các thuộc tính (một số thuộc tính của nút này cũng là nút – nằm trong nhóm nút **Geometric Properties Nodes**):

* **Coordinate**: là nút xác định một tập hợp các điểm ảnh trong không gian (đây là các điểm đỉnh của các đường thẳng).
* **Color**: nút này nhằm xác định màu sắc tại các điểm ảnh trong nút **Coordinate**. Số thành phần trong nút này có thể không nhất thiết phải bằng với số thành phần trong nút **Coordinate**.
* **coordIndex**: trường này bao gồm một dãy chỉ số thứ tự cho các điểm ảnh tạo nên đường thẳng. Ví dụ *coordIndex [0 1 2 0]* có nghĩa là đường thẳng được tạo bởi điểm thứ nhất nối với điểm thứ hai, điểm thứ hai nối với điểm thứ ba, điểm thứ ba nối với điểm thứ nhất, các điểm ảnh được xác định trong nút **Coordinate**. Một ví dụ khác *coordIndex [0 1 -1 2 0]*, ở đây kí hiệu “-1” cho biết hình ảnh được tạo trong VRML gồm hai đường thẳng: một đường thẳng tạo bởi điểm thứ nhất nối với điểm thứ hai và một đường thẳng khác tạo bởi điểm thứ ba nối với điểm thứ nhất.
* **colorIndex**: được sử dụng để chỉ định màu sắc cho các điểm ảnh (khi **colorPerVertex** có giá trị là true) hay cho các đường thẳng (khi **colorPerVertex** có giá trị là false) trong **coordIndex**. Trường này lấy màu sắc được quy định từ nút **Color**. Ví dụ *colorIndex [0 1 0 1]*trong ví dụ đầu tiên (*coordIndex [0 1 2 0]* và **colorPerVertex TRUE**) ở trên có nghĩa là xác định tại màu sắc thứ nhất cho điểm thứ nhất và điểm thứ ba, màu sắc thứ hai cho điểm thứ hai và thứ tư, còn trong ví dụ thứ hai (*coordIndex [0 1 -1 2 0]* và **colorPerVertex FALSE**) thì có nghĩa là đường thẳng nối hai điểm thứ nhất và thứ hai sẽ có màu thứ nhất, đường thẳng nối hai điểm thứ tư và thứ năm sẽ có màu sắc thứ hai.
* **ColorPerVertex**: có giá trị boolean. Khi **colorPerVertex** nhận giá trị TRUE thì màu của đường thẳng sẽ là màu trung bình của hai màu tại hai điểm tạo nên đường thẳng. Và khi trường này nhận giá trị là FALSE thì các đường thẳng đều có màu đơn lẻ hoặc màu mặc định (trắng).

Ví dụ:

Shape{

appearance Appearance{}

geometry IndexedLineSet{

                        coord Coordinate{

                                    point [3.0 0.0 0.0, 0.0 4.0 0.0, 0.0 0.0 5.0]

                        }

color Color{

color [1.0 0.0 0.0, 0.0 0.0 1.0]

}

colorIndex [0 1 0 1]

coordIndex [0 1 2 0]

colorPerVertex TRUE

}

}

### 2.6.3 Xây dựng khung bề mặt trong không gian

Nút **IndexedFaceSet** sẽ tạo nên bề mặt bằng cách kết hợp các điểm với nhau.

Cú pháp:

IndexedFaceSet{

            coord Coordinate{}

            coordIndex[]

            color Color{}

            colorIndex[]

            colorPerVertex TRUE

            convex TRUE

            solid TRUE

}

Các tham số:

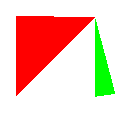
* **Coordinate**, **coordIndex**, **Color**, **colorIndex**, **colorPerVertex**: có đặc điểm tương tự như trong nút **IndexedLineSet** đã nói ở trên. Chú ý là bề mặt luôn luôn được xác định bởi các đường thẳng khép kín vì thế không cần chỉ ra điểm đầu tiên lại một lần nữa trong trường **coordIndex**.

Ví dụ:

**coordIndex [0 1 2 3]**

Nếu có bốn điểm xác định, khi liên kết các điểm này sẽ tạo ra hình tứ giác bằng cách sử dụng thuộc tính **coordIndex**[] (coordIndex[0 1 2 3]) có nghĩa là điểm thứ nhất nối với điểm thứ hai, điểm thứ hai nối với điểm thứ ba, điểm thứ ba nối với điểm thứ tư và điểm thứ tư nối với điểm thứ nhất.

* **Convex**: định nghĩa bề mặt là lồi hay lõm, trường này có giá trị kiểu boolean. Bộ trình duyệt VRML chỉ vẽ bề mặt lồi, trong trường hợp vẽ bề mặt lõm, bộ trình duyệt chia bề mặt đó thành các bề mặt lồi bé hơn để vẽ.
* **solid**:sử dụng để cho phép các mặt phẳng tạo ra có thể nhìn từ hai phía mặt phẳng hoặc không. Ví dụ *solid FALSE* tức là cho phép nhìn mặt phẳng ở hai phía, *solid TRUE* thì chỉ cho phép nhìn mặt phẳng ở một phía.

Ví dụ:

**Shape** **{**

**appearance** Appearance{}

**geometry** **IndexedFaceSet** **{**

**coord** **Coordinate** **{**

**point** [0 0 0,1 1 0,0 1 0,1 1 0,1 0 0,1 0 1**]**

**}**

**coordIndex** [0 1 2 **-**1 3 4 5**]**

**color** **Color** **{**

**color** [1 1 1,1 0 0**]**

**}**

**colorIndex** [1 0**]**

**colorPerVertex** **FALSE**

**convex** **FALSE**

**solid** **FALSE**

**}**

**}**

**2.6.4 Xây dựng khung lưới và bản đồ trong không gian**

Nút **ElevationGrip** cho phép xây dựng khung lưới được tạo bởi các điểm có độ cao xác định trong không gian. Thẻ này rất hữu ích cho việc xây dựng các mạng lưới hoặc địa hình chẳng hạn chúng ta có thể sử dụng nút này để vẽ một bản đồ địa hình trong không gian ảo.

Hình ảnh xây dựng được đặt trong không gian của mặt phẳng OXYZ. Điểm bắt đầu là gốc tọa độ, các điểm còn lại tạo nên lưới nằm theo hướng dương của các trục OX và OZ.

Cú pháp:

ElevationGrid {

  xDimension X

  xSpacing Y

  zDimension Z

  zSpacing W

  height []

  color Color []

  colorPerVertex TRUE

  convex TRUE

  solid TRUE

}

Các tham số:

* **Color**, **colorPerVertex**, **convex**, **solid**: có tính chất tương tự như đã nói ở phần trên, **solid** ở đây nếu bằng TRUE thì sẽ không cho phép nhìn từ dưới lên đối với mạng lưới tạo ra.
* **xDimension**: chứa số điểm trong mạng lưới nằm trên trục X.
* **zDimension**: chứa số điểm bên trong lưới nằm trên trục Z.
* **xSpacing**: là khoảng cách của hai điểm liên tiếp nhau theo hướng của trục X.
* **zSpacing**: khoảng cách của hai điểm liên tiếp nhau theo hướng của trục Z.
* **height**: chứa một danh sách các giá trị độ cao của mỗi điểm trong lưới.

Các điểm này được tính theo thứ tự từ trái sang phải từ trên xuống dưới.

Ví dụ:

Shape {

            appearance Appearance {

                        material Material {}

            }

            geometry ElevationGrid {

                        xDimension 9

                        zDimension 9

                        xSpacing 1.0

            xSpacing 1.0

                        solid TRUE

                        height [

                                  0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0,

                                  0.0, 1.0, 0.0, 0.0, 1.0, 0.0, 1.0, 1.0, 0.0,

                                  0.0, 0.0, 2.0, 0.0, 2.0, 2.0, 2.0, 0.0, 0.0,

                                  0.0, 0.0, 0.0, 3.0, 3.0, 3.0, 1.0, 0.0, 0.0,

                                  0.0, 1.0, 2.0, 3.0, 3.5, 3.0, 2.0, 1.0, 0.0,

                                  0.0, 0.0, 1.0, 3.0, 3.0, 3.0, 0.0, 0.0, 0.0,

                                  0.0, 0.0, 2.0, 2.0, 2.0, 0.0, 2.0, 0.0, 0.0,

                                  0.0, 1.0, 1.0, 1.0, 1.0, 0.0, 0.0, 1.0, 0.0,

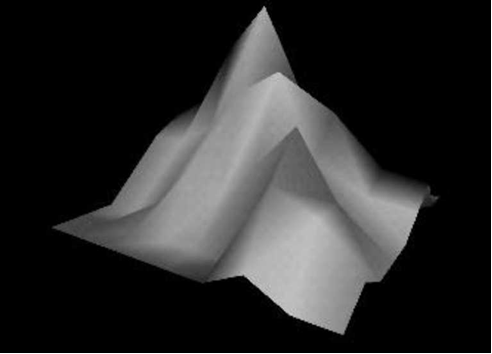
                                  0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0,

                        ]

            }

}

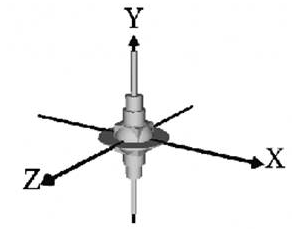
Kết quả của đoạn code trên:



*Hình 2.1 Ví dụ về khung lưới và bản đồ trong không gian*

## 2.7 Các phép biến đổi trong VRML

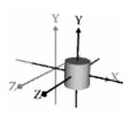
### 2.7.1 Transform

Mặc định, mọi vật thể và hình ảnh được xây dựng trong VRML đều được đặt ở tọa độ gốc và các phép biến đổi được dùng để tạo ra một hệ thống hệ trục tọa độ mới mà hệ trục này có vị trí tương đối so với hệ tọa độ mặc định. Có các phép biến đổi như: dịch chuyển, co giãn, quay. Ngoài ra, nếu các hình này được xây dựng trùng lên nhau như hình 2.2 có thể giúp chúng ta tạo ra một hình phức tạp mà không cần phải dùng đến nút **ElevationGrip**.

*Hình 2.2 Một hình phức tạp được xây dựng trong hệ trục XYZ.*

Nút Transform quản lý các phép biến đổi trong VRML như: dịch chuyển, co dãn, quay. Nút Transform gồm có các trường con: translation (dịch chuyển), rotation (quay), scale (co giãn), children. Trong đó trường children được dùng để chứa các đối tượng (vật thể, hình ảnh,...).

### 2.7.2 Phép dịch chuyển (Translation)

Cú pháp:

            Transform {

                        translation X Y Z

                        children []

}

Tham số:

* **translation** X Y Z: dịch chuyển gốc tọa độ đến điểm X Y Z.

### 2.7.3 Phép quay (Rotation)

Cú pháp:

            Transform {

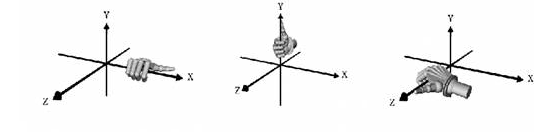
                        rotation X Y Z G

                        children []

}

Tham số:

* **rotation** X Y Z G: quay trục tọa độ theo trục X, Y, Z một góc G.
* G được tính theo radians.

Lưu ý: quy định, chiều quay đối tượng được xác định bởi quy tắc bàn tay phải.

*Hình 2.3 Quay theo trục X, Y, Z*

(Nắm bàn tay phải lại, choãi ngón cái ra theo như hình vẽ. Ngón tay cái chỉ phương của trục quay thì chiều từ cổ tay đến các ngón tay chi chiều dương khi quay đối tượng).

### 2.7.4 Phép co giãn (Scale)

Cú pháp:

            Transform {

                        scale x y z

                        children []

}

Tham số:

* **scale** X Y Z: co giãn đối tượng theo trục X, Y, Z với tỉ lệ x, y, z.

### 2.7.5 Children

Children: Các đối tượng được vẽ trong trường này sẽ chịu ảnh hưởng của các phép biến đổi trong nút Transform.

Ví dụ:

Transform {

            translation 2.0 1.0 0.0

            scale 2.0 1.0 3.0

            rotation 0.0 0.0 1.0 0.79

            children [

                        Shape {

                        appearance Appearance {

                                    material Material { }

                        }

                        geometry Cylinder {

                                    radius 0.3

                                    height 6.0

                                    top TRUE

                        }

                        }

            ]

}

Shape {

            appearance Appearance {

                        material Material { }

            }

            geometry Box { size 1.0 1.0 1.0}

}

Như trong ví dụ này thì hệ tọa độ của hình trụ sẽ dịch chuyển đến điểm (2, 1, 0) so với hệ tọa độ gốc, sau đó kích thước của hình trụ sẽ tăng 2 lần theo chiều trục X, tăng 3 lần theo chiều trục Z, và quay 0.79 radians theo trục Z. Còn hình hộp thì nằm tại tọa độ gốc.

## 2.8 Màu sắc và hình ảnh trên đối tượng trong VRML

Các đối tượng trong thế giới ảo khi được tạo ra sẽ có màu mặc định là màu trắng nhưng chúng ta có thể thay đổi chúng nhờ vào nút Appearance được cung cấp sẵn trong VRML, nút này được sử dụng để quy định các thành phần xuất hiện trên đối tượng như màu sắc, hình ảnh, video,… Nút Appearance bao gồm các nút con như: Material, ImageTexture, MovieTexture…

### 2.8.1 Màu sắc trong VRML

Nút **Material** có chức năng tạo màu cho các đối tượng trong VRML. Màu sắc trong VRML được thể hiện qua 3 tham số là R G B (Red-Green-Blue) với giá trị nằm trong khoảng từ 0.0 đến 1.0. Nút **Material** có các thuộc tính sau:

* **diffuseColor**: xác định màu sắc của đối tượng.
* **emissiveColor**: xác định màu của ánh sáng phát ra từ đối tượng.
* **transparency**: xác định độ trong suốt của đối tượng (0: hiển thị rõ, 1: trong suốt).
* **ambientIntensity**: xác định lượng phản chiếu ánh sáng của đối tượng.
* **specularColor**: xác định màu sắc của các điểm sáng bóng trên bề mặt của đối tượng.
* **shininess**: điều chỉnh cường độ sáng cho những điểm sáng bóng.

Ví dụ:

Shape {

            appearance Appearance {

                        material Material {

                        diffuseColor 1.0 0.5 0.0

                        emissiveColor 0.0 0.0 0.0

                        transparency 0.5

                        specularColor 0.5 0.5 0.5

                        shininess 0.3

                        ambientIntensity 0.4

                        }

            }

            geometry Box { size 1.0 1.0 1.0}

}

### 2.8.2 Vẽ hình trên đối tượng

Ngoài việc tô màu cho một đối tượng nào đó, trong VRML còn cho phép ta vẽ hình trên bề mặt đối tượng đó, hình vẽ được tạo ra từ tập hợp các điểm ảnh. Nút PixelTexture sẽ giúp ta thực hiện công việc này.

Ví dụ:

Shape {

appearance Appearance {

                        material Material {}

                        texture PixelTexture {

image 2 4 3 0xFF0000 0xFF00 0 0 0 0 0xFFFFFF 0xFFFF00

repeatS TRUE

repeatT TRUE

                        }

}

            geometry Cylinder {height 5}

}

Các thuộc tính:

* **image**: xây dựng một hình vẽ từ tập hợp các điểm ảnh.
* **repeatS và repeatT**: hình được lập đi lập lại theo chiều dọc và chiều ngang.

Ở trong ví dụ trên ta vẽ một bức hình gồm 8 điểm ảnh (mỗi điểm có các màu sắc được chỉ ra bởi các số thập lục phân trong ví dụ ở trên), sau đó bức hình được kéo giãn ra bằng với kích thước của hình trụ và dán lên hình trụ đó.

### 2.8.3 Dán hình lên đối tượng

Thay vì phải mất thời gian để vẽ hình lên đối tượng như trên ta có thể lấy một bức ảnh có sẵn dán lên bề mặt đối tượng như là dán decal. Kĩ thuật này được gọi là ánh xạ kết cấu (texture mapping).

Trong VRML có thể sử dụng một trong các định dạng ảnh sau đây:

* **GIF**: 8bit, có hỗ trợ trong suốt, nhưng không phải là sự lựa chọn tốt cho “texture mapping” vì hình ảnh sẽ bị “bể” khi dán lên đối tượng.
* **JPEG**: từ 8-16 bit, không hỗ trợ trong suốt, là sự lựa chọn tốt cho “texture mapping”.
* **PNG**: từ 8-16 bit, hỗ trợ trong suốt trên từng điểm ảnh, là sự lựa chon tốt nhất cho “texture mapping”.

Ví dụ:

Shape {

  appearance Appearance {

            material Material {}

            texture ImageTexture {

url “hinha.jpg”

repeatS TRUE

repeatT TRUE

}

  }

  geometry Box { size 3 4 5}

}

Ở ví dụ này chúng ta lấy hình ảnh “hinha.jpg” từ bên ngoài thông qua thuộc tính **url** dán lên đối tượng, các thuộc tính **repeatS** và **repeatT** tương tự như nút **PixelTexture**.

### 2.8.4 Chiếu phim trên bề mặt đối tượng

Ngoài việc cho phép vẽ hay dán hình lên đối tượng, VRML còn hỗ trợ việc chiếu một tập tin video trên đối tượng.

VRML chỉ hỗ trợ định dạng video MPEG (bao gồm MPEG1-Systems (âm thanh và hình ảnh) và MPEG1-Video (chỉ có hình ảnh)).

Cú pháp:

MovieTexture {

            loop      FALSE

            speed      1.0

startTime    0

            stopTime    0

            url       []

            repeatS     TRUE

            repeatT     TRUE

}

Các thuộc tính:

* **loop**: xác định tính lập lại của bộ phim.
* **speed**: xác định tốc độ chiếu phim, ví dụ speed có giá trị là 2 thì tốc độ chiếu phim sẽ nhanh gấp hai lần.
* **startTime và stopTime**: xác định khoảng thời gian chiếu và dừng phim dựa trên thời gian thực.
* **url**: tập các liên kết đến các tập tin video định trình chiếu.

Ví dụ:

Shape {

            appearance Appearance {

                        material Material {}

                        texture MovieTexture {

                        url “phim.mp4”

                        loop TRUE

                        }

            }

            geometry Cylinder {height 5}

}

## 2.9 Âm thanh, ánh sáng, camera, phong cảnh trong VRML

Để cho thế giới ảo được thực hơn thì trong thế giới đó không thể thiếu các thành phần như phong cảnh, âm thanh, ánh sáng và cả các góc quan sát (do camera tạo ra). VRML hỗ trợ một số nút để chúng ta có thể đưa các thành phần nói trên vào thế giới ảo.

### Âm thanh

Để làm cho thế giới ảo trở nên sinh động và hấp dẫn hơn, chúng ta nên thêm âm thanh vào đó. Ta có thể tạo ra âm thanh nền, tiếng chuông cửa, âm thanh khi mở cửa hoặc bất cứ âm thanh nào mà mình muốn có trong thế giới ảo. Tất cả những điều đó được thực hiện bởi hai nút do VRML cung cấp là nút Sound và nút AudioClip. Có thể hiểu đơn giản như sau: nút Sound được dùng để xác định vị trí phát ra âm thanh trong thế giới ảo và âm thanh đó được xác định bởi nút AudioClip.

#### Sound

VRML không chỉ hỗ trợ âm thanh 2D mà còn hỗ trợ cả âm thanh 3D nên khi chúng ta xác định vị trí phát ra âm thanh thì không chỉ đơn thuần là chúng ta xác định vị trí mà còn là xác định cả không gian truyền âm của âm thanh đó.

Cú pháp:

Sound {

source NULL

location 0 0 0

intensity 1

direction 0 0 1

priority 0

spatialize TRUE

minBack 1

minFront 1

maxBack 10

maxFront 10

}

Các thuộc tính:

* **source**: xác định cụ thể là một nút AudioClip hoặc một nút MovieTexture chứa âm thanh phát ra.
* **location**: xác định vị trí phát của nguồn âm thanh.
* **intensity**: cường độ âm thanh phát ra, giá trị nằm trong khoảng [0, 1] với 1 là cường độ âm thanh tối đa.
* **direction**: hướng đi của âm thanh trong không gian, giá trị là một vectơ ba chiều.
* **priority**: độ ưu tiên của âm thanh, được sử dụng khi có nhiều âm thanh trong cùng một thế giới, giá trị được xác định từ 0 đến 1, giá trị càng cao tức độ ưu tiên càng cao.
* **spatialize**: nếu có giá trị là TRUE thì âm thanh phát ra sẽ là 3D nhưng âm thanh sẽ bị đồng bộ hóa, lúc đó bạn có thể nghe thấy âm thanh trong cả hai kênh trái và phải nếu bạn đứng bất kì vị trí nào trong vùng phát thanh.
* **minBack và minFront**: xác định một vùng ellipsoid xung quanh vị trí phát âm thanh, nếu ở trong vùng này thì âm thanh nghe được sẽ bằng với cường độ âm thanh do thuộc tính intensity quy định.
* **maxBack và maxFront**: xác định một vùng ellipsoid bao ngoài vùng ellipsoid trên, nếu ở ngoài vùng này thì sẽ không nghe được âm thanh nguồn, từ khoảng vùng ellipsoid trong đến vung ellipsoid ngoài âm thanh sẽ giảm dần.

Ví dụ:

Sound {

      source AudioClip {

            url    “theme2.wav”

            loop   TRUE

     }

     minFront 40.0

     minBack  40.0

     maxFront 80.0

     maxBack  80.0

     location 19.5 7.0 -30.0

   }

1. **AudioClip**

Nút AudioClip được sử dụng để xác định đường dẫn và thuộc tính tập tin nhạc. Nút này chỉ hỗ trợ định tập tin MIDI hoặc WAVE.

Cú pháp:

AudioClip {

loop FALSE

pitch 1.0

startTime 0

stopTime 0

url [ ]

description ““

}

Các thuộc tính:

* **loop**: xác định tính lặp lại của âm thanh.
* **pitch**: tốc độ phát âm thanh, ví dụ có giá trị là 2 thì âm thanh phát ra sẽ nhanh hơn gấp 2 lần.
* **startTime và stopTime**: xác định thời gian phát và dừng của âm thanh.
* **url**: chỉ ra tập các đường dẫn đến các tập tin nhạc.
* **description**: ghi chú, được dùng để mô tả các âm thanh.

Ví dụ:

AudioClip {

            url    “theme2.wav”

            loop   TRUE

            pitch   2

            description   “Âm thanh nền”

}

### 2.9.2 Ánh sáng

Khi quan sát một vật thể bất kỳ, các vật thể đó chỉ được chiếu sáng bởi một ánh sáng xung quanh. Đây là một ánh sáng đặc biệt được tạo ra bởi trình duyệt. Tuy nhiên, chúng ta cũng có thể điều khiển ánh sáng này thông qua nút **NavigationInfo**, đây là một nút được sử dụng để xác định các thuộc tính của người sử dụng. Tuy nhiên để có được một thế giới ảo trông thật hơn thì chúng ta nên tạo ra các nguồn sáng như trong thế giới thật, ví dụ như: ánh sáng mặt trời, ánh sáng của đèn điện,… Và VRML hỗ trợ cho chúng ta các loại ánh sáng đó thông các nút **DirectionalLight**, **PointLight**, **SpotLight**.

#### a, NavigationInfo

Nút này được sử để mô tả người sử dụng và các mô hình điều hướng. Nút này gồm các trường thuộc tính sau:

* **avatarSize**: xác định kích thước vật lý của người xem với mục đích nhằm phát hiện các va chạm với vật thể cùng di chuyển trên địa hình trong thế giới ảo. Giá trị của trường này là một tập ba giá trị, ví dụ: [0.25 1.6 0.75], giá trị đầu tiên chỉ ra khoảng cách tối thiểu giữa người dùng với vật thể (khi khoảng cách giữa người dùng và đối tượng bằng giá trị này thì xem như người dùng chạm vật thể), giá trị thứ hai xác định chiều cao của người dùng và giá trị cuối cùng chỉ ra chiều cao tối đa của vật mà người dùng có thể bước qua, ví dụ với cầu thang, khoảng cách về chiều cao của các bậc thang phải nhỏ hơn hoặc bằng giá trị này thì người dùng mới có thể bước lên được.
* **headlight**: xác định là có (TRUE) hoặc không (FALSE) ánh sáng đặc biệt như đã nói ở trên.
* **speed**: tốc độ di chuyển của người sử dụng.
* **type**: kiểu di chuyển của đối tượng, ví dụ “FLY” tức người dùng sẽ di chuyển theo kiểu bay.
* **visibilityLimit**: xác định khoảng cách tối đa mà người dùng có thể nhìn thấy được vật thể, mặc định có giá trị là 0 tức khoảng cách này là vô hạn.

Ví dụ:

NavigationInfo {

avatarSize [0.25, 1.6, 0.75]

headlight TRUE

speed 1.0

type “WALK”

visibilityLimit 0.0

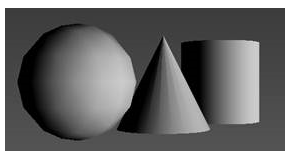
}

#### b, DirectionalLight

Nút **DirectionalLight** xác định một nguồn sáng được đặt từ rất xa so với thế giới của chúng ta (nguồn sáng này không có vị trí xác định). Ánh sáng của nó bao gồm nhiều tia sáng song song với nhau và có hướng do chúng ta quy định (được quy định bởi một vectơ 3D), nút này được sử để tạo ra các loại ánh sáng giống như ánh sáng mặt trời trong thế giới ảo.

Các thuộc tính:

* **on**: quy định việc hoạt động của nguồn sáng, giá trị có kiểu boolean.
* **intensity**: xác định cường độ chiếu sáng của nguồn sáng.
* **ambientIntensity**: xác định bao nhiêu ánh sáng này góp phần vào việc chiếu sáng tổng thể cả thế giới. Giá trị nằm trong khoảng [0, 1].
* **color**: màu sắc của nguồn sáng.
* **direction**: chỉ ra hướng chiếu sáng của các tia sáng. Giá trị của trường này là một vectơ 3D.

Ví dụ:

DirectionalLight {

on TRUE

intensity 1

ambientIntensity 0

color 1 1 1

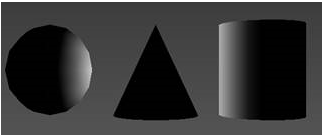
direction 0 0 -1

}

#### c, PointLight

Nút này xác định một nguồn sáng tại một vị trí xác định, các tia sáng của nguồn sáng này đi theo mọi hướng, không hướng về một phương cụ thể nào, nguồn sáng này giống như ánh sáng của bóng đèn điện trong thế giới thực của chúng ta. Và cũng giống như ánh sáng của bóng đèn điện, nguồn sáng này bị giới hạn về khoảng cách chiếu sáng. Vì vậy nút **PointLight** thường được sử dụng trong việc tạo ra các vật thể phát sáng như bóng đèn.

Ví dụ:

PointLight {

on TRUE

intensity 1

ambientIntensity 0

color 1 1 1

location 0 0 0

attenuation 1 0 0

radius 100

}

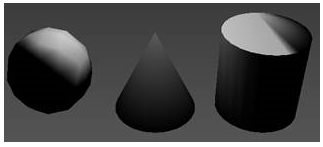
Các thuộc tính:

* Các thuộc tính **on**, **intensity**, **ambientIntensity**, **color** tương tự như các thuộc tính của nút **DirectionLight**.
* **location**: vị trí của nguồn sáng, có giá trị là một điểm trong không gian.
* **attenuation**: có giá trị là một vectơ 3D xác định việc tăng giảm cường độ cũng như khoảng cách chiếu sáng của nguồn sáng. Giá trị của trường này phải khác (0, 0, 0).
* **radius**: chỉ ra phạm vi chiếu sáng của nguồn sáng, giá trị này phải lớn hơn 0.

#### d, SpotLight

Nút này xác định một nguồn sáng tương tự như nút **PointLight** nhưng nguồn sáng này có các tia sáng phát ra theo các hướng tạo thành dạng hình nón (các tia sáng có đỉnh trùng nhau và bị giới hạn bởi một hình nón có đỉnh là đỉnh các tia sáng), tức nguồn sáng này chỉ chiếu sáng một khu vực chỉ định. Nút SpotLight được sử dụng để tạo ra các vật thể phát sáng như đèn pha của xe máy.

Ví dụ:

SpotLight {

on TRUE

intensity 1

ambientIntensity 0

color 1 1 1

location 0 0 0

direction 0 0 0

attenuation 1 0 0

radius 100

cutOffAngle 0.78

beamWidth 1.57

}

Các thuộc tính của nút SpotLight:

* Các thuộc tính **on**, **intensity**, **ambientIntensity**, **color, location, attenuation, radius** tương tự như nút **PointLight**.
* **cutOffAngle**: quy định các hình nón hạn chế các tia sáng, phải có giá trị lớn hơn hoặc bằng 0 và nhỏ hơn 1,57 radian (tương đương với 180 độ).
* **beamWidth**: xác định một hình nón bên trong hình nón của trường **cutOffAngle**, các tia sáng trong hình nón này có cường độ đồng nhất. trường này phải có giá trị lớn hơn hoặc bằng 0 và nhỏ hơn 1,57 radian (tương đương với 180 độ). Các tia sáng nằm ngoài hình nón của thuộc tính **beamWidth** và nằm trong hình nón của thuộc tính **cutOffAngle** có cường độ giảm dần từ trong ra ngoài.

### 2.9.3  Camera

Khi ta mở một tập tin VRML thì trình duyệt sẽ hỗ trợ mặc định cho chúng ta một góc nhìn, để tùy biến góc nhìn đó ta có thể sử dụng nút **Viewpoint**. Nút **Viewpoint** có chức năng xác định vị trí của người dùng trong thế giới và các thông số của góc nhìn. Với nút này chúng ta có thể tùy biến vị trí và các đặc tính quan sát của người dùng khi thế giới được khởi động lên, việc đó sẽ giúp cho người xem như đang ở trong thế giới thực. Nút này có các thuộc tính như sau:

* **fieldOfView**: xác định một góc trong “radians”. Giá trị của trường này là một số thực nằm trong khoảng từ 0 đến 3,142 và có giá trị mặc định là 0,785398. Góc này sẽ tương đương với ống kính máy quay (camera) nếu có giá trị nhỏ và sẽ tương đương với ống kính góc rộng nếu có giá trị lớn (chẳng hạn có giá trị 3,14), tuy nhiên lúc đó góc nhìn sẽ bị méo.
* **position**: xác định vị trí của người dùng trong thế giới ảo. Có giá trị là một điểm trong không gian.
* **orientation**: Xác định biên độ quay của góc quan sát, trường này có giá trị là kiểu SFRotation (xem trong phần kiểu dữ liệu đã trình bày ở trên).
* **description**: cung cấp một đoạn mô tả cho góc quan sát.
* **jump**: cho phép sự chuyển giữa các góc nhìn trong thế giới, nếu có giá trị TRUE thì cho phép người di chuyển theo góc nhìn, còn nếu có giá trị là FALSE thì chỉ đơn thuần là sự chuyển đổi góc nhìn mà không hề ảnh hưởng đến vị trí của người dùng.

Ví dụ:

Viewpoint {

fieldOfView 0.785398

position 0 0 10

orientation 0 0 1 0

description “Góc nhìn số 1”

jump TRUE

}

### 2.9.4   Phong cảnh nền và môi trường

Việc xây dựng một thế giới ảo gồm tất cả các đối tượng giống như với thế giới thực là chuyện không thể vì ta không có đủ thời gian để thực hiện việc đó. Vì thế chúng ta chỉ có thể xây dựng một thế giới ảo giống một phần (khu vực) nào đó của thế giới thực. Với các khu vực xung quanh, nếu chúng ta để trống các khu vực này thì thế giới của chúng ta sẽ không giống như thực được. Đây thật sự là một vấn đề khó khăn, và VRML cung cấp cho chúng ta một công cụ giúp che phủ các phần còn lại đó để thế giới của chúng ta trông thực hơn. Nút **Background** được cung cấp nhằm mô tả các đường chân trời của bạn, nó cho phép chúng ta xác định bầu trời, mặt đất và các hình ảnh xung quanh thế giới của chúng ta.

#### a, Background

            Cú pháp:

Background {

skyColor [ ]

skyAngle [ ]

groundColor [ ]

groundAngle [ ]

backUrl [ ]

bottomUrl [ ]

leftUrl [ ]

rightUrl [ ]

frontUrl [ ]

topUrl [ ]

}

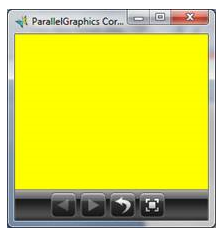
Các thuộc tính:

* **skyColor** và **skyAngle**: xác định các màu sắc trên bầu trời, nếu muốn bầu trời có màu sắc đơn lẻ thì chúng ta chỉ cần thiết lập các giá trị cho trường **skyColor** và bỏ trường **skyAngle** (hình a), còn nếu muốn bầu trời có hiệu ứng màu gradient (hình b) thì ta cần phải thiết lập các giá trị cho hai trường nói trên. Ví dụ ta sẽ thiết lập như sau:
* Trong hình a:

skyColor [1 1 0, .9 .9 .5, .7 .7 .9, 0 0.4 1, 0 0.7 1, 1 1 1]

* Trong hình b:

skyAngle [ 0.075, 0.1, 0.2, 1.309, 1.5708 ]skyColor [ 1 1 0, .9 .9 .5, .7 .7 .9, 0 0.4 1, 0 0.7 1, 1 1 1 ]



*Hình a*                                                            *Hình b*

* **groundColor** và **groundAngle**: cũng tương tự **skyColor** và **skyAngle**.
* **backUrl**, **bottomUrl**, **leftUrl**, **rightUrl**, **frontUrl**, **topUrl**: xác định các hình vẽ bao quang thế giới của chúng ta, nếu chúng ta thiết lập hai thuộc tính **topUrl** và **bottonUrl** thì chúng ta có thể bỏ các trường **groundColor**, **groundAngle**, **skyColor**, **skyAngle**.

#### b, Fog

Nút **Fog** được sử dụng nhằm tạo ra môi trường cho thế giới ảo để nó trở nên thực hơn. Nút này có chức năng cung cấp không khí hay tạo ra một màn sương dày tùy thuộc vào quy định của chúng ta. Nó bao gồm các thuộc tính như sau:

* **color**: màu sắc của làn sương.
* **fogType**: xác định mật độ của làn sương tùy vào khoảng cách. Thuộc tính này gồm hai giá trị “LINEAR” và “EXPONENTIAL”. Với giá trị “LINEAR” thì sương mù sẽ tăng dần một cách tuyến tính theo khoảng cách còn với giá trị “EXPONENTIAL” thì làn sương sẽ như một môi trường tự nhiên.
* **visibilityRange**: xác định khoảng cách tối đa có thể nhìn thấy vật trong làn sương mù. Nếu có giá trị là 0 thì có nghĩa là không có làn sương.

Ví dụ:

Fog {

color 1 1 1

fogType “LINEAR”

visibilityRange 1

}

## 2.10 Nhóm các đối tượng trong VRML

Có thể nhóm các đối tượng đơn lẻ trong VRML thành một đối tượng phức tạp. Việc đặt một nhóm đối tượng này vào nhóm đối tượng kia tạo thành một lược đồ cấu trúc các nút. Một nút trong nhóm có thể có nhiều nút con đặt bên trong trường children.

VRML hỗ trợ các nút nhóm sau: Anchor, Billboard, Group, Switch, Transform, Inline.

### 2.10.1 Anchor

Anchor: nút này có chức năng xác định một tập hợp các đối tượng và gắn một siêu liên kết đến một url, chẳng hạn như là một siêu liên kết đến thế giới VRML khác, đến một trang HTML hoặc đến một dữ liệu nào đó mà trình duyệt có thể đọc được. Khi kích chuột vào một trong các đối tượng bên trong nút Anchor thì toàn bộ trình duyệt sẽ đưa đến địa chỉ url. Khi đưa chuột lên một đối tượng nào đó trong nút này thì có thể nhìn thấy được dòng ghi chú của nó. Tất cả các nút trong nút Anchor đều hiển thị.

Ví dụ:

Anchor {

children [

Shape {

geometry Sphere {}

}

]

url “http://ninkuhack.blogspot.com/”

description “Blog của tôi!”

parameter [“target=my\_frame” ]

bboxCenter 0 0 0

bboxSize -1 -1 -1

}

Các thuộc tính:

* **children**: chứa các nút con trong nhóm.
* **url**: xác định địa chỉ của siêu liên kết. Có thể có nhiều liên kết đặt trong url, khi đó bộ trình duyệt sẽ tìm kiếm dữ liệu trong các liên kết này theo thứ tự ưu tiên giảm dần.
* **parameter**: cung cấp thêm thông tin cho trình duyệt. Ví dụ ta có thể xác định cửa sổ mà url hiển thị. Như ví dụ trên là hiện url trên một cửa sổ khác.
* **description**: ghi chú, dòng ghi chú này sẽ hiển thị khi người dùng di chuột qua đối tượng chứa trong nút **Anchor**.
* **bboxCenter**: xác định tâm của một hình hộp bao quanh các nút trong nhóm. Giá trị của trường này là một điểm trong không gian.
* **bboxSize**: xác định kích thước của hình hộp trên bao quanh các nút trong nhóm. Mặc định trường này có giá trị -1 -1 -1 tức hình hộp không được xác định. Các giá trị thành phần của trường này phải lớn hơn hoặc bằng 0. Nếu các nút con không nằm trong hình hộp thì hình hộp là không xác định.

### 2.10.2 Group

**Group**: tạo ra một tập hợp các đối tượng như là một thực thể duy nhất. Nút này có các trường: **children**, **bboxCenter**, **bboxSize**. Các trường này tương tự như của nút **Anchor**. Tất cả các nút con nằm trong nút **Group** đều hiển thị.

Ví dụ:

Group {

children […]

bboxCenter 0 0 0

bboxSize -1 -1 -1

}

### 2.10.3 Switch

Switch: tạo ra một nhóm chuyển đổi gồm tập hợp các nút con, chỉ có một nút con trong nhóm được hiển thị và nút này là do người dùng lựa chọn. Các nút con được đánh số thứ tự từ 0, nếu trường **whichChoice** có giá trị -1 tức là không chọn nút con nào.

Ví dụ:

Switch {

whichChoice -1

choice […]

}

Các thuộc tính:

* **whichChoice**: lựa chọn một nút con trong nhóm.
* **Choice**: chứa tập các nút con trong nhóm.

### 2.10.4 Transform

Transform: tạo ra một nhóm các đối tượng và đặt chúng tại hệ trục tọa độ mới. Nút này đã được trình bày ở trên.

### 2.10.5 Billboard

**Billboard:** nút này tạo ra một nhóm các đối tượng và được đặt trên một hệ tọa độ đặc biệt. Hệ tọa độ này luôn được quay về đối diện với người dùng. Tất cả các nút trong nhóm đều được hiển thị.

Ví dụ:

Billboard {

children […]

axisOfRotation 0 1 0

bboxCenter 0 0 0

bboxSize -1 -1 -1

}

Các thuộc tính:

* **axisOfRotation**: xác định một vectơ 3D sử dụng trong việc quay hệ tọa độ. Nếu vectơ này có giá trị là (0, 0, 0) thì trục Z của hệ tọa đọ này sẽ hướng thẳng về người dùng.

**2.10.6 Inline**

**Inline**: tạo ra một nhóm các đối tượng đặc biệt được lấy ra từ một tập tin VRML khác (được chỉ ra sau **url**). Tất cả các đối tượng trong tập tin này sẽ được hiển thị. Nút này thường được sử dụng để gọi trực tiếp một đối tượng bên ngoài vào thế giới hiện tại.

* v  Ví dụ:

Inline {

url [“cuaso.wrl”]

bboxCenter 0 0 0

bboxSize -1 -1 -1

}

Các thuộc tính:

* **url**: chứa tập đường dẫn của các tập tin VRML ngoài thế giới hiện tại

## Tái sử dụng các đối tượng trong VRML

### 2.11.1 Inline

**Inline**: sử dụng nút **Inline** để gọi trực tiếp một đối tượng bên ngoài tập tin vào tập tin hiện tại. Nút này đã được trình bày ở trên.

Cú pháp:

Inline {url ”Tên đường dẫn”}

Vi dụ:

Inline {url ”robo.wrl”}

### DEF

**DEF** (DEFine): cho phép ta định nghĩa một đối tượng hay một kiểu thuộc tính.

Ví dụ:

·        Tạo đối tượng:

Shape {

appearance Appearance {

material DEF RedColor Material {

diffuseColor 1.0 0.0 0.0

                        }

            }

            geometry Box { size 5 5 5}

}

·        Sử dụng:

Shape {

            appearance Appearance {

                        material USE RedColor

            }

            geometry Sphere { }

}

Như ví dụ trên ta định nghĩa đối tượng RedColor kiểu **Material**. Sau đó muốn sử dụng lại ta chỉ gọi hàm bằng từ khóa **USE**.

### 2.11.3 PROTO

**PROTO** (PROTOTYPE): cho phép định nghĩa một đối tượng có kèm theo tham số truyền vào.

Ví dụ:

·        Tạo đối tượng:

PROTO MyBox [field SFColor mausac 1 0 0

                        field SFVec3f kthuoc 1 1 1]

{

            Shape {

                        appearance Appearance {

                                    material Material {

                                    diffuseColor IS mausac

                                    }

                        }

                        geometry Box { size IS kthuoc}

            }

}

·        Sử dụng:

MyBox **{** mausac 0 1 0

      kthuoc 5 5 5**}**

            Trong ví dụ này ta thấy ta đã định nghĩa một đối tượng “MyBox” với hai tham số truyền vào là màu sắc và kích thước của đối tượng là một hình hộp có màu sắc và kích thước là chưa xác định, sau này nếu muốn tạo ra một đối tượng có dạng hình hộp ta chỉ cần gọi tên đối tượng đã tạo (ở đây là “MyBox”) cùng với các giá trị thuộc tính của nó (ở đây là màu sắc và kích thước).

# CHƯƠNG 3: ỨNG DỤNG VRML - XÂY DỰNG CHƯƠNG TRÌNH MÔ PHỎNG TÒA NHÀ A12 ĐẠI HỌC CÔNG NGHIỆP HÀ NỘI

## 3.1 Yêu cầu bài toán

Trước thực tế hiện nay, cùng với sự phát triển công nghiệp hóa – hiện đại hóa của đất nước, đã và đang mọc lên hàng loạt các công trình xây dựng, đặc biệt là các tòa nhà lớn. Bởi vậy việc mô phỏng chúng sẽ rất cần thiết. Vậy thì họ làm thế nào để mô phỏng nó?

## 3.2 Khảo sát và phân tích

Nhóm tới trường thực hiện khảo sát thực tế tòa nhà A5 đại học Công nghiệp Hà Nội. Phân tích các đối tượng có trong khuôn viên tòa nhà A5, lấy số liệu để thực hiện mô phỏng sau này



*Hình 3.1 Ảnh toà nhà A5 ĐHCN HN*

Sau khi phân tích, nhóm chia thành các đối tượng chính:

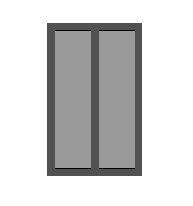
|  |  |
| --- | --- |
| **Đối tượng chính** | **Yêu cầu** |
| Banner nhà A5 | Có hình dáng cơ bản, giống tương đối |
| Bậc thang trước cửa | Có hình dáng cơ bản, giống tương đối |
| Biển nhà A5 | Có hình dáng cơ bản, giống tương đối |
| Cầu thang | Có hình dáng cơ bản, giống tương đối |
| Điều hòa | Có hình dáng cơ bản, giống tương đối |

## 3.3 Xây dựng chương trình mô phỏng

### 3.3.1 Cửa sổ sau

Mỗi một cánh là kết hợp của 2 thanh ngang dọc và tấm kính.

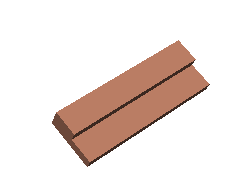
* Các thanh ngang dọc: sử dụng geometry box
* Tấm kính sử dụng transparency để tạo độ trong suốt



*Hình 3.2 cửa sổ sau A5*

### 3.3.2 Bậc thang trước cửa

Gồm 3 Geometry Box chồng lên nhau



*Hình 3.3 Bậc thang*

### 3.3.3 Biển nhà A5

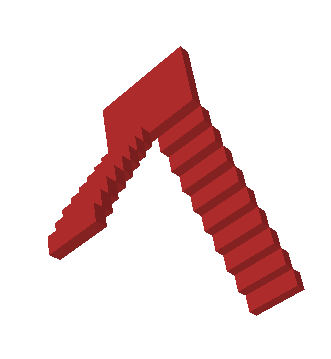
Gồm Geometry Box và Geometry Text



*Hình 3.4 Biển nhà A12*

### 3.3.4 Cầu thang

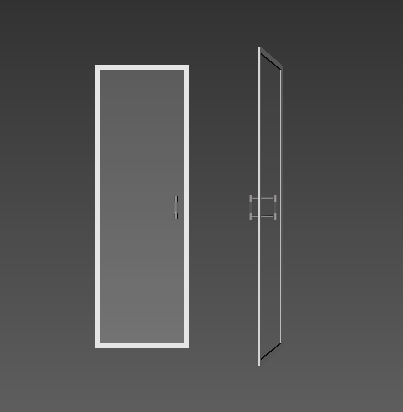
Gồm nhiều Geometry Box ghép lại



*Hình 3.5 Cầu thang*

### 3.3.5 Cửa ra vào

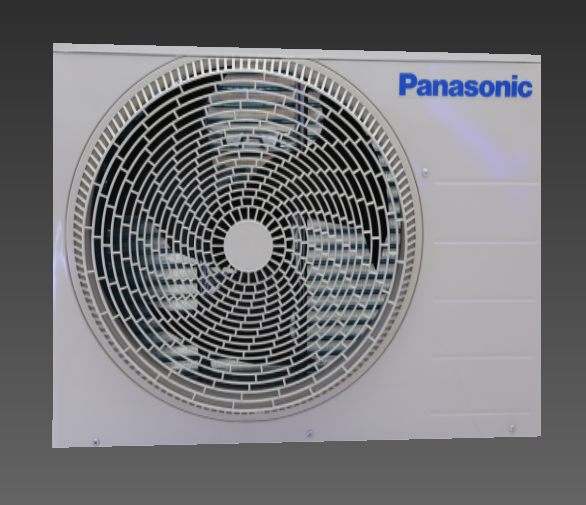
Gồm các Geometry Box ghép với nhau tạo thành khung cửa. Kính cũng là Geometry Box nhưng cho màu với độ trong suốt (transparency)



*Hình 3.9 Cửa ra vào*

### 3.3.6 Điều hòa

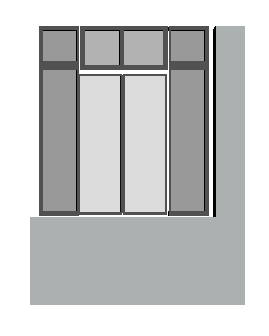
Gồm Geometry Box dán ảnh (Image Texture)



*Hình 3.11 Điều hòa*

*3.3.7 Cửa sổ trước*

Gồm các Geometry Box ghép với nhau tạo thành khung cửa. Kính cũng là Geometry Box nhưng cho màu với độ trong suốt (transparency)



### 

### 3.4 Toàn cảnh

Toàn cảnh được phân ra thành các phần nhỏ:

* Bầu trời và nền
* Background {
* skyColor [ 1 1 0, .9 .9 .5, .7 .7 .9, 0 0.4 1, 0 0.7 1, 1 1 1 ]
* skyAngle [ 0.075, 0.1, 0.2, 1.309, 1.5708 ]
* groundColor [ 1 1 0, .9 .9 .5, .7 .7 .9, 0 0.4 1, 0 0.7 1, 1 1 1 ]
* groundAngle [ 0.75, 1.4, 1.5, 1.5708 ]
* backUrl [ ]
* bottomUrl [ ]
* leftUrl [ ]
* rightUrl [ ]
* frontUrl [ ]
* topUrl [ ]
* }
* Ánh sáng
* # ánh sáng chiếu vào phía trước
* DirectionalLight {
* direction 0 0 1
* intensity 0.65
* #ambientIntensity 0.1
* color 1 1 1
* }
* # ánh sáng chiếu vào phía sau
* DirectionalLight {
* direction 0 0 -1
* intensity 0.3
* color 1 1 1
* }

## 3.5 Kết quả

Sau khi ghép nối các thành phần lại với nhau, ta được:

*Hình 3.15 Toàn cảnh nhà A5*



# KẾT LUẬN

Quaquá trình tìm hiểu và thực hiện đề tài: “***Mô phỏng tòa nhà A5 ĐHCN HN***“, nhóm chúng em đã học hỏi thêm và củng cố được nhiều kiến thức về VRML và tính thiết thực của nó trong thực tế.

Về phần xây dựng chương trình mô phỏng, nhóm tự nhận thấy vẫn còn một số phần:

Ưu điểm:

* Về cơ bản đã hoàn thành việc mô phỏng một tòa nhà trong thực tế
* Mô phỏng được một số đối tượng
* Thực hiện được một số cảm biến

Hạn chế:

* Các đối tượng mô phỏng chưa được đẹp.
* Một số phần lấy từ trên mạng.

# TÀI LIỆU THAM KHẢO

* **Một số tài liệu:**

Giáo trình VRML – Vũ Đức Huy

* **Một số trang web:**

*https://www.google.com*