**Lời Cảm Ơn**

Lời đầu tiên tôi xin bày tỏ lòng biết ơn chân thành tới các thầy cô giáo Trường Đại Học Bách Khoa – Đại Học Đà Nẵng nói chung và các thầy cô bộ môn công nghệ phần mềm nói riêng. Trong suốt thời gian tôi học tại trường, các thầy cô đã luôn tận tình dạy dỗ, chỉ bảo để tôi có được kết quả như ngày hôm nay.

Đặc biệt, tôi xin chân thành cảm ơn giảng viên, tiến sĩ Lê Thị Mỹ Hạnh. Cảm ơn cô vì những định hướng, nhận xét quý báu và động viên kịp thời để tôi hoàn thành khóa luận này. Tôi cũng cảm ơn Gameloft – Đà Nẵng đã tạo điều kiện cho tôi có môi trường làm việc trong quá trình hoàn thành khóa luận này.

Cuối cùng tôi xin bày tỏ lòng biết ơn sâu sắc tới gia đình, bạn bè, những người đã luôn động viên, giúp đỡ tôi cả về vật chất lẫn tinh thần trong suốt những năm tháng qua.

Đà nẵng, ngày …. Tháng … năm…

Sinh viên

DANH MỤC CÁC TỪ VIẾT TẮT

Bounding box Hình hộp bé nhất chứa trọn vẹn các đối tượng 3D

Mesh : Các đa giác dùng để mô tả hình dạng của vật thể trong không gian 3 chiều

Local space: không gian riêng của đối tượng 3D

Clip space : không gian tọa độ vertex được chiếu lên bởi camera

Cube map: là tập hợp 6 texture 2D được xếp lien tục trong 6 mặt của hình hộp

Shader : là tập các chỉ lệnh cho GPU được dùng để Render các hiệu ứng.

Texture: Các hình ảnh được dán lên các đối tượng 3D nhằm tăng độ chi tiết của đối tượng

Vertex : Đỉnh trong không gian 3D

Vertex program : chương trình shader xử lí các biến đổi với vertex

World space : không gian thế giới trong OpenGL

Fragment program : chương trình shader xử lí các biến đổi với mỗi pixel.

Vertex buffer : vùng đệm chứa các vertex.

NDC : Normalized Device Coordinates (Hệ trục tọa độ thiết bị)

NDK: Native Development Kit

JVM : Java Virtual Machine

JNI (Java Native Interface)

**CHƯƠNG 1 – CHƯƠNG MỞ ĐẦU**

* 1. **Lý do chọn đề tài**

Ngành công nghiệp phát triển game trên thế giới đang phát triển như vũ bão. Cách đây 20 năm, game rất đơn giản và chỉ một người hoặc một nhóm nhỏ người phát triển nhưng ngày nay, các game hiện đại với đồ họa và khả năng tương tác ấn tượng thường được phát triển bởi một đội ngũ đông đảo người thiết kế game, lập trình viên, nghệ sĩ… trong thời gian ròng rã từ một đến ba năm. Chính việc game ngày càng trở nên phức tạp như vậy nên hiện nay, các game thương mại không còn phát triển từ nguyên thủy nữa mà được phát triển lên từ game engine. Game engine hỗ trợ việc xây dựng game một cách nhanh chóng, đơn giản hơn đồng thời cung cấp khả năng tái sử dụng code cao do đó có thể phát triển nhiều game từ một game engine.

Hiện nay hầu hết các nhà phát triển game lớn và nổi tiếng như Vavle, Squa-Enix, Ubisoft, Gameloft…hoặc các nhóm phát triển game nhỏ. thường xây dựng các engine cho riêng họ. Có một vài lý do mà một công ty game hay một nhóm phát triển game (studio) lựa chọn xây dựng thay vì mua công nghệ. Các studio này thường cần những yêu cầu cụ thể mà các phần mềm trung gian chưa đáp ứng được, việc tự xây dựng các chức năng này thường tốn ít thời gian hơn việc chờ các phần mềm trung gian đáp ứng các yêu cầu này. Ngoài ra các lập trình viên trong các nhóm phát triển game do không hiểu đầy đủ về công nghệ mà họ không xây dựng cũng phải rất cảnh giác trước nhiều lỗi không mong muốn. Thêm nữa các nhóm phát triển nhỏ có thể không đủ chi phí hoặc nghĩa vụ hợp đồng trả cho các phần mềm trung gian.

Tôi đã từng thử qua một vài engine phổ biến cho game mobile hiện nay như Unity vấn đề là tôi không hoàn toàn hiểu hết những gì xảy ra đằng sau engine. Tôi muốn có một sự hiểu biết sâu về những gì mà engine đang thực hiện và có thể thao tác với nó một cách tốt nhất. Với mong muốn đó cùng với việc bùng nỗ game trên nền tảng di động, tôi chọn xây dựng cho mình một game engine xây dựng một game và triển khai ra trên hai nền tảng window và di động sử dụng ngôn ngữ C++ ( một ngôn ngữ rất tốt cho việc xây dựng game đa nền tảng ).

Tôi chọn đề tài tốt nghiệp này không phải với một tham vọng quả lớn, chỉ là muốn đi những bước chân chập chững đầu tiên vào thế giới phát triển game rộng lơn, để thu lượm kiến thức về lĩnh vực khó khăn nhưng đầy thú vị này và hy vọng ở một tương lai không xa, tôi có thể góp một phần sức lực vào ngành công nghiệp game của nước nhà.

* 1. **Mục Tiêu và Nhiệm vụ**

**\_Mục tiêu:** Tìm hiểu được cấu trúc của một game engine, xây đựng được một game engine cho mình, và từ đó phát triển một game 3D triển khai trên nền tảng PC (Window) và mobile (Android).

**\_Nhiệm vụ:**

* Tìm hiều lập trình OpenGL ES 2.0
* Tìm hiểu cấu trúc 1 game
* Thiết kế mô hình hệ thống game
* Xây dựng game engine
* Thiết kế game Kill The Rat mang tính giải trí đơn thuần
* Triển khai game sang nền tảng di động ( Android )
  1. **Đối tượng nghiên cứu và phạm vi hệ thống**

**Đối tượng:**

* Mô hình xây dựng hệ thống game engine đa nền tảng
* Hệ điều hành window và tài nguyên hệ thống Android

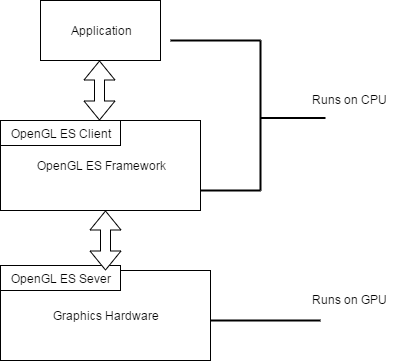
**Phạm vi:** Đề tài chỉ dừng lại tại phần xây dựng phần vẽ (render) cho game engine và triển khai game Kill The Rat trên nền tảng window và Android Mobile

**Chương 2: Cở Sở Lý Thuyết**

**2.1.** **Tổng quan về Game Engine**

* + 1. **Khái niệm Game Engine**

Game Engine là một công cụ hỗ trợ, một lớp trung gian ở giữa các ứng dụng game và nền tảng bên dưới, các thư viện lập trình cấp thấp. Game Enigine giúp phát triển ứng dụng game một cách nhanh chóng và đơn giản, đồng thời cung cấp khả năng tái sử dụng mã nguồn cao do có thể phát triển nhiều ứng dụng game khác nhau từ một game Engine.



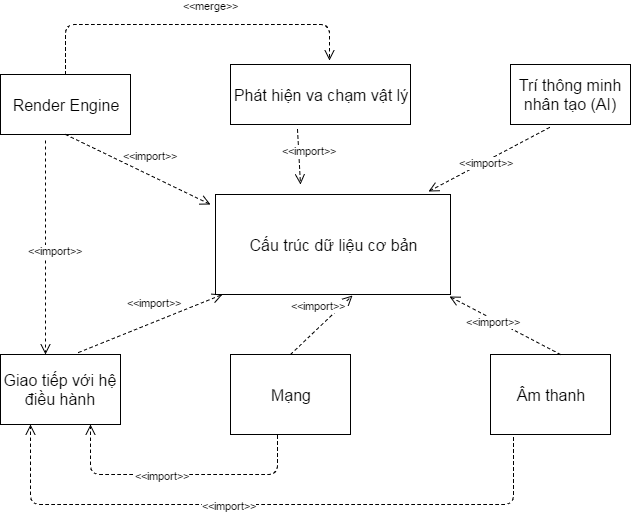
Hình 1: Kiến trúc tổng quan của Game Engine OpenGL ES

* + 1. **Mô hình Game Engine**

Game Engine hỗ trợ đa nền (Window PC và Mobile Android). Và nhắm đến thị trường Game Mobile đang rất phổ biến trên các thiết bị di động.

Đầu tiên là khả năng chạy đa nền. Để đạt được điều này, Engine sử dụng các bản build khác nhau trên các nền tảng khác nhau (chứ không phải sử dụng thông dịch). Để giải quyết vấn đề này, đầu tiên em chọn build game bằng Visual Studio 2013 trên nền tảng window sử dụng C++. Sau đó sử dụng NDK trên android để sử dụng code C++ và build game ra Mobilde Android.

Engine được thiết kế hướng đối tượng và yêu cầu hiệu năng chạy cao nên việc sử dụng ngôn ngữ C++ ngôn ngữ đáp ứng điều kiện trên.

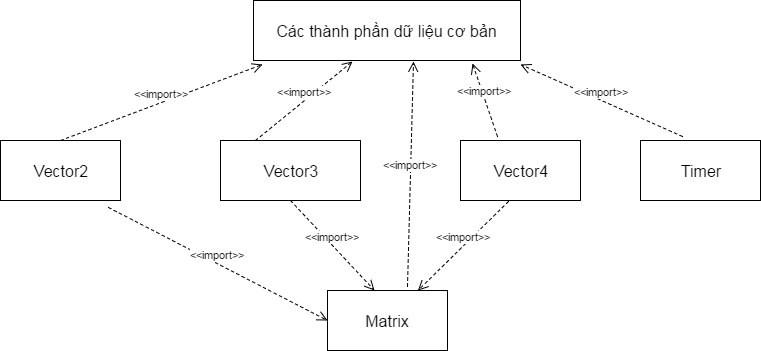


Kiến trúc tổng thể của Game Enigne

Do giới hạn về mặt thời gian, nên phạm vị luận văn này tôi tập trung vào các thành phần sau: Các kiểu dữ liệu cơ bản, Render Engine. Những thành phần này đủ để hỗ trợ người sử dụng tạo ra một game 3D và tương tác với chúng.

1. Thành phần cấu trúc dữ liệu cơ bản

Các kiểu dữ liệu cơ bản là thành phần bao gồm các cấu trúc dữ liệu cơ bản như mãng động, vector, ma trận…, và các phép toán trên các kiểu dữ liệu đó; cung cấp cho các thành phần khác sử dung.



Hinh . Biểu đồ lớp của các thành phần dữ liệu cơ bản

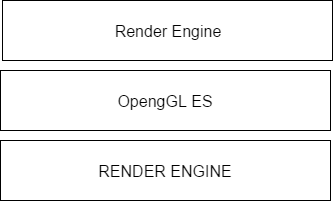
Vector2, Vector3, Vector4: Các loại vector biểu diễn tọa độ 2 chiều, 3 chiều và tọa độ đồng nhất.

Matrix: ma trận được sử dụng để biểu diễn các phép biến đổi: dịch, xoay, co giãn trong không gian 3 chiều.

1. Thành phần Render Enginer

Render Engine là thành phần cốt lõi của một Game Engine. Nó hỗ trợ người dùng các công việc thiết yếu để tạo ra một khung cảnh 3D. Người dùng sẽ không cần biết nhiều đến những công việc tầng thấp như quá trình đọc file tài nguyên, sủ dụng các API đồ họa 3D, tạo các hiệu ứng… mà quản lý logic các đối tượng bằng các lớp mà Render Engine cung cấp.

Đây là thành phần duy nhất trong Engine giao tiếp với phần cứng đồ họa thông qua thư viện đồ họa 3D cấp thấp, cụ thể ở đây là OpenGL ES.



Hinh Kiến trúc phân tầng của Render Engine

Thành phần này được thiết kế thành các mô đun như sau:

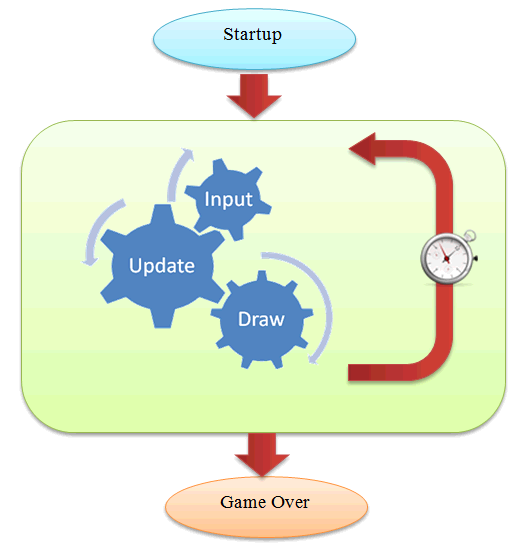
* Resource Manager là mô đun quản lý tài nguyên cần thiết để xây dựng nên các Object và cảnh 3D như texture 2D , cubemap… Mô đun này giúp cho việc dễ dàng quản lí tài nguyên , tăng tốc độ game và tránh việc sử dụng các model , texture trùng nhau
* Scene Manager là mô đun quản lý các màn chơi trong game (Scene), bao gồm các hàm chuyển màn chơi, pause ..

Chuyển động của đối tượng là mô đun mở rộng các thành phần tương tác giữa các object trong môi trường.

3.1 Vòng lặp game

Phần cốt lõi của hầu hết các game chính là vòng lặp được dùng để cập nhật và hiển thị trạng thái của game.

Minh họa:



* 1. **Tổng quan về Opengl**

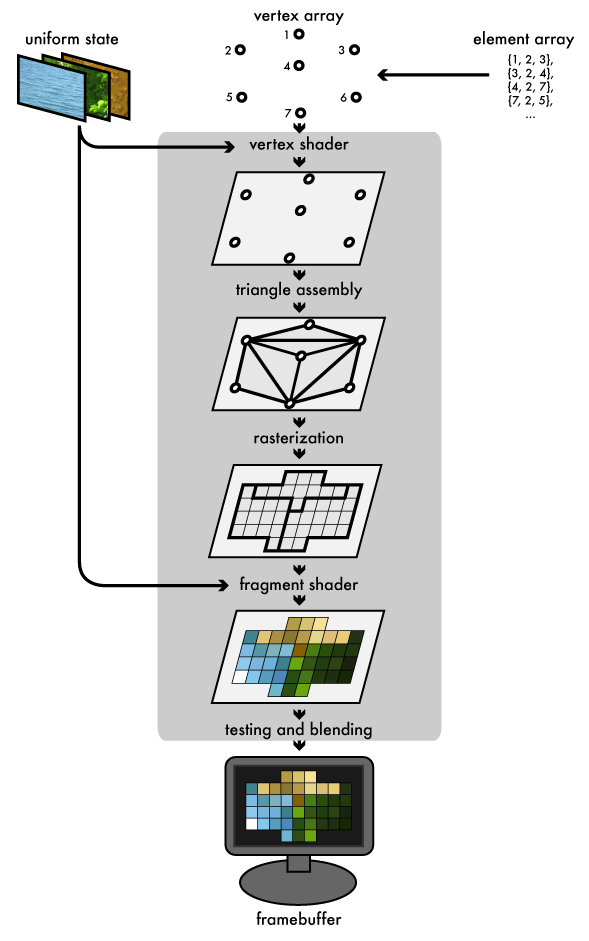
OpenGL là 1 crossplatform API, dùng để render 2D và 3D graphics. API tương tác trực tiếp với GPU (nhân xử lí đồ họa) để tăng tốc render cho phần cứng.

OpenGL định nghĩa, cung cấp các function để draw 2D và 3D Graphics, bao gồm khoảng 250 functions khác nhau.

Hiện nay OpenGL đã phát triển đến phiên bản 4.5, hỗ trợ cho các dòng GPU mới nhất.

* 1. Rendering pipeline

Đây là 1 khái niệm căn bản trong 3D Graphics, cần hiểu rõ trước khi viết bất kì chương trình 3D nào  
Renderring pipeline/Graphics pipeline là 1 quá trình gồm 1 chuỗi các bước để tạo ra ảnh bitmap(2D) của 1 scene 3D. Chúng ta biết rằng hình ảnh hiển thị trên màn hình thực chất là 1 ảnh 2 chiều gồm nhiều pixels, tương ứng với số pixels của màn hình (vd 1920x1080 pixels), màn hình có 1 tần số quét(vd 60hz) tương ứng với số lần refresh trong 1s, trong mỗi lần refresh thì ảnh bitmap trên màn hình được cập nhật và chúng ta có được hình ảnh liên tục không bị ngắt quãng.  
hình ảnh sau cho chúng ta graphics pipeline của OpenGL :



* + 1. Vertex and Element Arrays

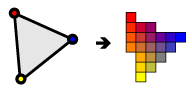
Việc dựng hình bắt đầu thông qua đường ống (pipeline) với một tập hợp của một hoăc nhiều vùng đệm đỉnh ( vertex buffer), được điền đầy với các mảng thuộc tính của vertex. Các thuộc tính này được sử dụng như đầu vào cho vertex shader. Các thuộc tính đỉnh phổ biến bao gồm vị trí của đỉnh trong không gian 3 chiều, màu sắc…Tập các bộ đệm đỉnh cung cấp dữ liệu cho một công việc dựng hình được gọi chung là mảng đỉnh (Vertex array). Khi việc render được thực thi, chúng cung cấp một phần tử bổ sung, một mảng các chỉ mục (index) vào mảng đỉnh để xác định đỉnh nào được nạp vào đường ống. Thứ tự các chỉ mục cũng kiểm soát các đỉnh được lắp ráp vào tam giác sau này như thế nào.

3.3.2. Vertex shader

GPU bắt đầu bằng việc đọc các đỉnh được chọn từ mảng đỉnh (Vertex array) và chay thông qua vertex shader. Vertex shader là một chương trình lấy một tập các thuộc tỉnh của đỉnh làm đầu vào và đưa ra một tập các thuộc tính mới được gọi là giá trị varying, các giá trị này được nạp vào quá trình rasterizer. Vertex shader tạo ra các giá trị đầu ra varying khác nhau như màu săc, tọa độ textures giúp cho việc trộn lẫn các bề mặt tam giác trong quá trình kết nối đỉnh.

* + 1. Rasterization

Rasterization lấy mỗi tam giác, cắt và loại bỏ các phần bên ngoài màn hình và chia các phần còn lại có thể nhìn thấy thành các mảng có kích thước pixel. (fragments). Rasterizer sẽ pha trộn những màu sắc trên bề mặt như thể hiện trong sơ đồ sau



3.4.5. Fragment shader

Các đoạn phân mảnh (fragments) được tạo ra sau đó sẽ được đi qua một chương trình gọi là fragment shader. Chương trình này sẽ nhân các giá trị varying từ vertex shader được nội suy từ rasterizer làm giá trị đầu vào. Đầu ra của chương trình là các giá trị màu sắc, độ sâu mà sau đó được vẽ vào bộ đệm khung. Các thao tác fragment shader phổ biến bao gồm ánh xạ và kết cấu ánh sang.

* 1. Các phép biến hình (Transformations)

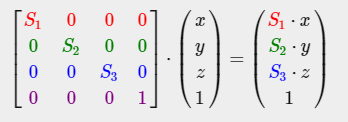
Để di chuyển các đối tượng (Object) trong môi trường 3D chúng ta có thể di chuyển bằng cách thay đổi đỉnh của chúng và định lại bộ đệm trong mỗi khung hình (frame) nhưng điều này khá là phức tạp và chi phí cho việc này là khá cao. Có nhiều cách tốt hơn để biển đổi một đối tượng đó là sử dụng cách nhân các ma trận.

* + 1. Co Dãn (Scaling)

Khi co dãn một vector, ta sẽ tăng độ dài vector và giữ nguyên hướng của chúng.

Trong môi trường 3D ứng với trường hợp 2D ở trên tương ứng với trục z là 1.

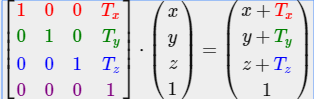
Để biển đổi vector trục tọa độ 3D ta định nghĩa một vector S(S1,S2,S3) sau đó định nghĩa một ma trận biển đổi tỉ lệ (Scale Matrix) cho vector như sau:



3.4.2. Tịnh tiến (Translation)

Translation là quá trình di chuyển vector bằng viêc cộng thêm một vector vào vector gốc.

ta định nghĩa một vector S(S1,S2,S3) sau đó định nghĩa một ma trận dich chuyển (Translation Matrix) cho vector như sau:

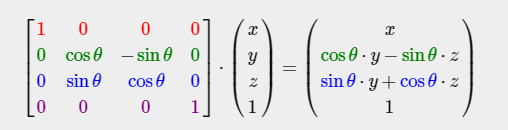


3.4.5 Xoay (Rotate)

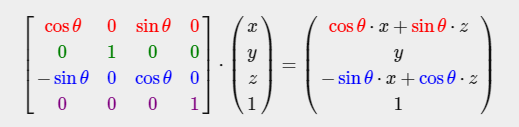
Xoay (Rotation) trong hệ trục tọa độ 3D được xác định bởi một góc quay và trục quay.

Ta sẽ sử dụng ma trận Rotate được tạo bằng cách sau để xoay quanh các trục tọa độ.

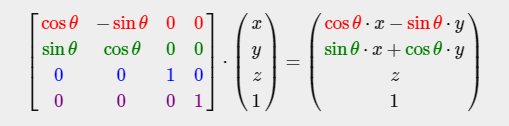
Xoay quanh trục X:



Xoay quanh trục Y:



Xoay quanh trục Z:



3.4.6 Kết hợp các phép biến hình

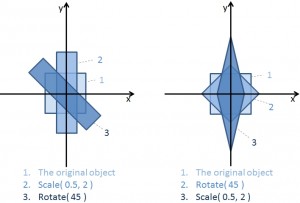
Nếu ta muốn kết hợp nhiều phép biến đổi trên cùng một vector, ta phải nhân các ma trận biến đổi lại theo một thứ tự nhất định để không thay đổi hình dạng ban đầu của vector.

Các ma trận phải được nhân theo thứ tự như sau:

Scale – Rotation - Translation

trong đó Rotation = Rz \* Rx \* Ry.

Dưới đây là ví dụ về hình dạng của đối tượng theo các thứ tự nhân ma trận khác nhau:



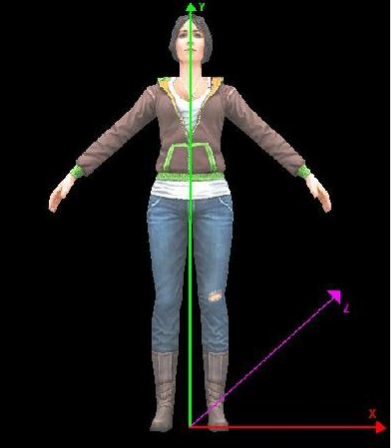
3.4.6. Hệ thống các trục tọa độ (Coordinate System)

Việc chuyển đổi các trục tọa độ tới tọa độ của thiết bị (NDC) và tọa độ màn hình thường được thực hiện theo từng bước. Trong đó ta chuyển tọa độ đỉnh của một đối tượng sang một vài hệ tọa độ trước khi chuyển chúng thành tọa độ màn hình. Ưu điểm của việc chuyển đổi sang một số hệ tọa độ trung gian là một số phép tính toán dễ dàng hơn trong các hệ tọa độ nhất định sẽ trở nên rõ ràng hơn.

Để chuyển đổi trục tọa độ từ một không gian (space) sang một trục toa độ không gian khác ta sử dụng một số các ma trận chuyển đổi, trong đó quan trọng nhất là ma trận mô hình (Model), hướng nhìn( View) và chiếu (Projection). Sau đây là một cái nhìn tổng quan về việc biến đổi thông qua các ma trận này.

 Hinh . Biến đổi góc nhìn từ hệ tọa độ mô hình sang hệ tọa độ màn hình

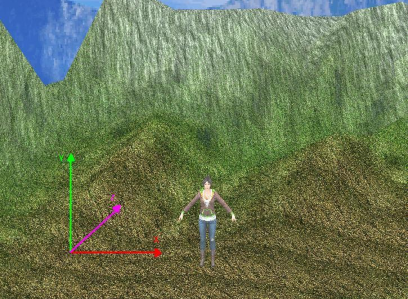
* Hệ tọa độ cục bộ (local space) là hệ tọa độ của đối tượng . Nơi mà các đối tượng (Object) được tạo ra.



Hinh Hệ trục tọa độ cục bộ (local space)

* Đối tượng từ hệ tọa độ cục bộ được chuyển sang hệ tọa độ thế giới ta bằng cách:

World Matrix \* Object



Hinh Đối tượng được đặt trong hệ tọa độ thế giới (World Space)

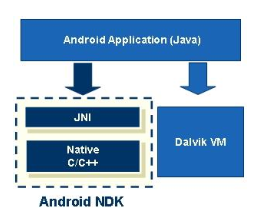
* Tiếp theo là quá trình biến đổi từ hệ tọa độ thế giới sang hệ tọa độ góc nhìn theo cách mà tọa độ này được nhìn thấy thông qua các camera hoặc từ điểm quan sát của người xem.
* Sau đó các tọa độ trong khung nhìn (view space) sẽ được co dãn sang trong pham vị -1.0 đến 1.0 để xác định xem đỉnh nào sẽ được xuất hiện trên màn hình
* -Cuối cùng ta biến đổi thành tọa độ màn hình trong quá trình biển đổi khung nhìn. Biến đổi các tọa độ từ -1.0 và 1.0 sang phạm vi tọa độ được xác định bởi glViewPort. Kết quá sau đó sẽ được gửi đến các rasterizer để biến chúng thành các mảnh (fragment).

3.4. Tổng quan về Android NDK

3.4.1 Android NDK là gì?

Andoroid NDK là một bộ công cụ cho phép thực thì các thành phần của ứng dụng sử dụng ngôn ngữ thuần (Native) như C hoặc C++.

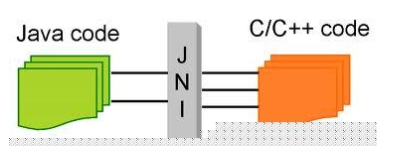
Android NDK giúp cho việc tái sử dụng mã nguồn C++, tăng hiệu năng ứng dụng , đồng thời giúp cho việc truy cấp vào hệ thống mà không phải thông qua máy ảo JVM



Hinh Các thành phần của Android NDK

3.4.2. JNI là gì?

JNI (Java Native Interface) là một framework cho phép gọi các hàm Java trong JVM từ các ngôn ngữ cấp thấp như C, C++ hay assembly. Nói nôm na dễ hiểu là khi ta muốn gọi Java từ C++ thì mình sẽ gọi thông qua JNI.



Hình Ứng dụng của JNI

CHương 3 Phân tích thiết kế hệ thống

3.1 Kiến trúc tổng thể của Render Engine

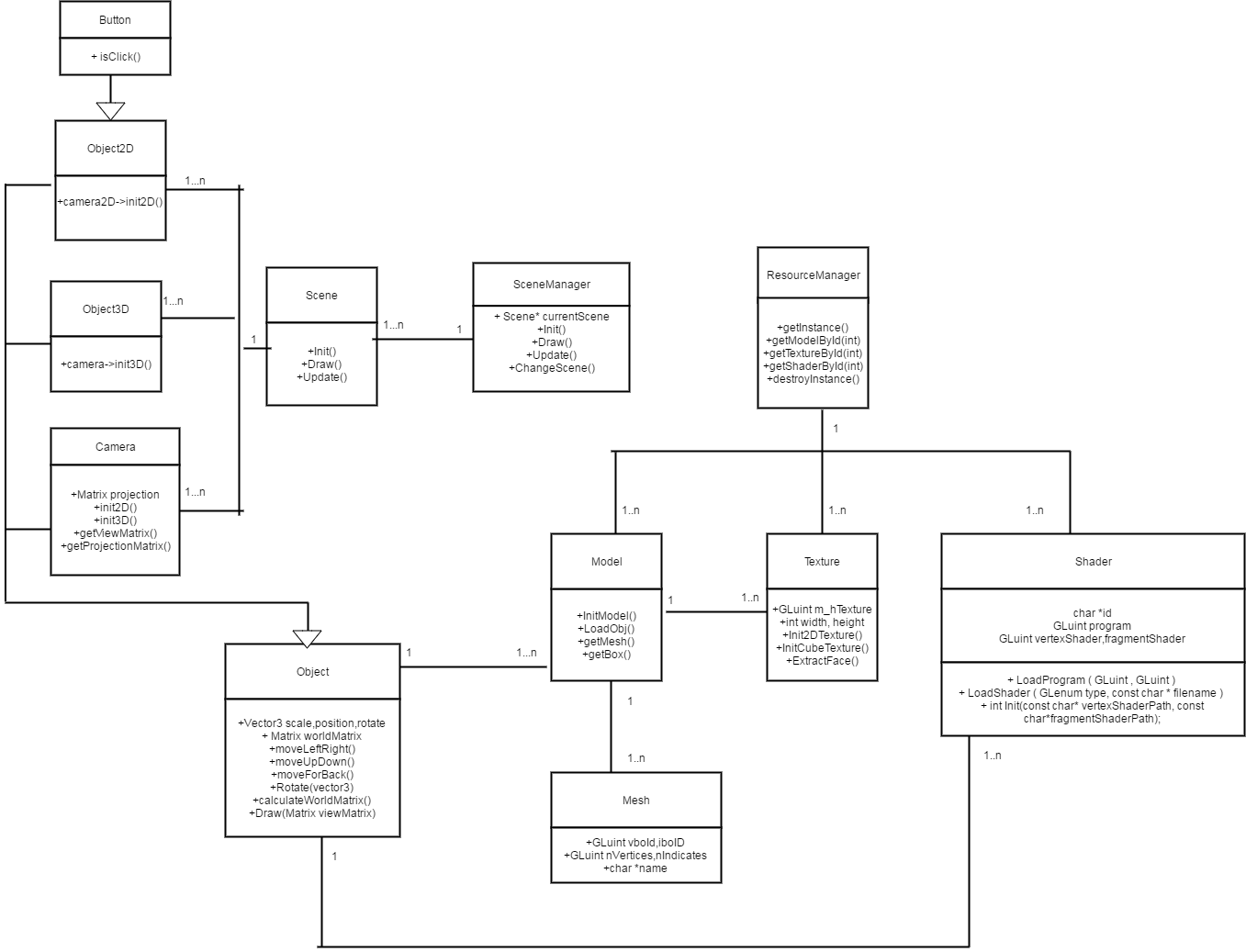
Render Engine được chia thành các mô đun chính sau:

1. **Các thành phần chính của Engine**

Engine hỗ trợ người dùng các công việc thiết yếu để tao cảnh 3D và các mô hình 3D. Người dùng sẽ không cần biết nhiều đến những công việc tầng thấp như quá trình đọc file tài nguyên, sử dụng các API đồ họa 3D, tạo các hiệu ứng… mà quản lý logic các đối tượng bằng các giao diện do Engine cung cấp.

Engine giao tiếp với phần cứng đồ họa thông qua thư viện đồ họa 3D cấp thấp, cụ thể ở đây là OpenGL .

Sau đây là biểu đồ lớp của engine



Hình 1.2 : OpenGL ES 2.0 Game Engine Class Diagram

* Resource Manager là mô đun quản lý tài nguyên cần thiết để xây dựng nên các Object và cảnh 3D như texture 2D , cubemap… Mô đun này giúp cho việc dễ dàng quản lí tài nguyên , tăng tốc độ game và tránh việc sử dụng các model , texture trùng nhau
* Scene Manager là mô đun quản lý các màn chơi trong game (Scene), bao gồm các hàm chuyển màn chơi, pause ..
* Scene là một cảnh game level, chẳng hạn như các cảnh trong 1 game thường có chia làm Start Scene, Play Scene , GameOver Scene. Việc chia làm các Scene và thông quan Scene Manager giúp cho việc quản lý game trở nên dễ dàng hơn, dễ control hơn trong quá trình phát triển game.
* Object chia làm 2 loại là Object 2D và Object3D. Việc mô phỏng chuyển động của các Object này sẽ thông qua một ma trận worldMatrix , dựa vào các vector postion, scale, rotate mà ta sẽ mô phỏng chuyển động cho các Object này. Chúng ta sẽ bàn kỹ hơn về vấn đề này ở chương sau.
* Camera trong engine giống như góc nhìn ngoài đời thật. Ở các vị trí khác nhau sẽ cho ra các kết quả khác nhau khi nhìn các Object. Camera cũng là một object trong engine , và bằng vào việc di chuyển camera trong môi trường 3D sẽ cho ra các kết quả khác nhau khi nhìn vào một cảnh game.
* Một Object sẽ bao gồm nhiều model và texture và mỗi model sẽ được tạo ra từ nhiều mesh.
* Shader sẽ giúp chúng ta load các file shader tạo hiệu ứng màu sắc cho object. Bằng việc này chúng ta chỉ việc thao tác với các file shader để tạo hiệu ứng cho object một cách đơn giản hơn.