Índice general

CAPÍTULO 1

Rendering

1.1. Introducción

Rendering es un término en inglés que denota el proceso de generar una imagen 2D a partir de un modelo digital 3D o un conjunto de ellos, a los que se les llama "escena". Puede ser comparado a tomar una foto o filmar una escena en la vida real.
FALTA DECIR ALGO MÁS...

1.2. ISGL3D

ISGL3D es un *framework* (marco de trabajo) para *iPad*, *iPhone* y *iPod touch* escrito en *Objective-C*, que sirve para crear escenas y *renderizarlas* de forma sencilla. Es un proyecto en código abierto y gratis. En su sitio web oficial: `www.isgl3d.com`, se puede descargar su código, y de forma sencilla ISGL3D puede ser agregado como un complemento de *Xcode*. Además se pueden encontrar tutoriales, una *Application Programming Interface* (API) y un acceso a un grupo de *google* donde la comunidad pregunta y responde dudas propias y ajenas. Una buena manera de iniciarse en manejo de la herramienta es siguiendo los tutoriales en: `www.isgl3d.com/resources/tutorials`; al menos este fué el camino elegido por el grupo. Los tutoriales son 6, y abarcan distintos tópicos:

- **Tutorial 0:** primer paso en el creado de una aplicación ISGL3D. Cubre algunos conceptos básicos y muestra cómo integrar la herramienta a *Xcode*.
- **Tutorial 1:** mustra cómo crear una escena bien simple, con tan sólo un cubo en rotación continua.
- **Tutorial 2:** enseña cómo agregar luz a una escena. Se ven las distintas fuentes de luz que existen en el *framework*.
- **Tutorial 3:** se ve cómo hacer para mapear texturas en los objetos 3D con el objetivo de hacrlos más realistas.
- **Tutorial 4:** muestra cómo crear interacción entre el usuario y los distintos objetos ISGL3D, cuando este los toca a través de la pantalla.
- **Tutorial 5:** se ven algunas nuevas primitivas y se muestra cómo *renderizar* objetos con transparencia.

Cuando se descarga e instala ISGL3D, se puede ver que la herramienta incluye un proyecto *Xcode* integrado por varios ejemplos para ejecutar y a la vez ver su código, otra buena forma de aprender cómo realizar distintas tareas de interés. Entre los ejemplos se encuentra cada uno de los tutoriales realizados adecuadamente.

A continuación se comentarán algunas características y conceptos de ISGL3D que fueron importantes para el proyecto; por detalles de algunos temas en particular referirse a la ya mencionada API en: `www.isgl3d.com/resources/api`.

Cuando se crea una aplicación ISGL3D, el núcleo de la misma es la llamada "*view*" ("vista" en Español). Una *view* esta compuesta principalmente por una escena y una cámara:

- Una **escena** (Isgl3dScene3D) es donde los objetos o modelos 3D son agregados como nodos. Todos los nodos pueden ser tanto trasladados como rotados y pueden tener otros nodos hijos; los nodos hijos son trasladados y rotados con sus padres. Así como objetos 3D, se pueden agregar luces de distinto tipo, que generarán en la escena efectos de sombra que luego serán adecuadamente *renderizados* en función de dónde se encuentre y hacia dónde este mirando la cámara.

- Una **cámara** es utilizada para para visualizar la escena desde una posición y un ángulo en particular. La cámara se manipula como cualquier otro objeto o nodo en la escena, se puede trasladar, rotar y hasta indicar hacia dónde quiere uno que la cámara apunte. Es importante ajustar la cámara de manera que su arquitectura sea la que uno busca. Se pueden entonces ajustar ciertos parámetros intrínsecos a esta como por ejemplo su campo visual, su distancia focal, la altura y la anchura del plano imagen, etc.

Es importante entender que el llamado *render* se realiza sumando la información de la escena, objetos 3D y sus hijos, luces, etc.; más la información de dónde se encuentra la cámara, sus características y hacia dónde esta apunta.

Un particularidad de la cámara de ISGL3D es que el parámetro intrínseco "distancia focal" visto la sección **??**, no es directamente configurable. En cambio, el valor que sí se puede alterar es el llamado "FOV", acónimo de "Fiel Of View". El *field of view* de una cámara no es más que su campo visual, y se mide como la extensión angular máxima mapeable en el plano imagen, medida desde el centro óptico C. Puede ser medido de forma horizontal o de forma vertical; sin embargo, en ISGL3D es definido verticalmente. Ver figura 1.1.

Realizando algo de geometría se ve que la relación entre la distancia focal y el FOV es:

$$FOV = 2.arctg\left(\frac{h}{2.f}\right)$$

donde h denota la altura del plano imagen y f la distancia focal de la cámara.

ISGL3D cuanta con algunas estructuras primitivas que pueden ser usadas como modelos, o incluso combinadas de manera de formar modelos algo más complejos. Las principales estructuras primitivas de ISGL3D son:

- **Isgl3DArrow:** modelo correspondiente a una flecha. Tiene 4 parámetros configurables:

  - *headHeight:* altura de la punta.
  - *headRadius:* radio de la punta.

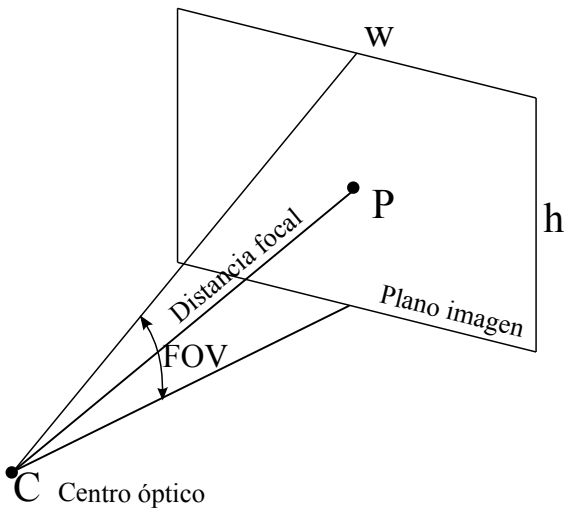

Figura 1.1: Definición gráfica del FOV.

- *height:* altura total de la flecha.
- *radius:* radio de la base.

- **Isgl3DCone:** modelo correspondiente a un cono. Tiene 3 parámetros configurables:
  - *bottomRadius:* radio de la base inferior.
  - *height:* altura del cono.
  - *topRadius:* radio de la base superior.
- **Isgl3DCube:** modelo correspondiente a un cubo. Tiene 3 parámetros configurables:
  - *depth:* profundidad del cubo.
  - *height:* altura del cubo.
  - *width:* anchura del cubo.
- **Isgl3DCylinder:** modelo correspondiente a un cilindro. Tiene 3 parámetros configurables:
  - *height:* altura del cilindio.
  - *radius:* radio del cilindro.
  - *openEnded:* indica si el cilindro cuenta con sus extremos abiertos o no.
- **Isgl3DEllipsoid:** modelo correspondiente a una elipsoide. Cuenta con 3 parámetros configurables:
  - *radiusX:* radio de la elipsoide en la dirección *x*.
  - *radiusY:* radio de la elipsoide en la dirección *y*.
  - *radiusZ:* radio de la elipsoide en la dirección *z*.
- **Isgl3DOvoid:** modelo ovoide. Cuenta con 3 parámetros configurables:
  - *a:* radio del ovoide en la dirección *x*.
  - *b:* radio del ovoide en la dirección *y*.

  - *k:* factor que modifica la forma de la curva. Cuando toma el valor 0, el modelo se corresponde con el de una ellipsoide.

- **Isgl3DSphere:** modello correspondiente a una esfera. Tiene un único parámetro configurable:
  - *radius:* radio de la esfera.
- **Isgl3DTorus:** modelo correspondiente a un toroide. Cuenta con 2 parámetros configurables:
  - *radius:* radio desde el origen del toroide hasta el centro del tubo.
  - *tubeRadius:* radio del tubo del toroide.

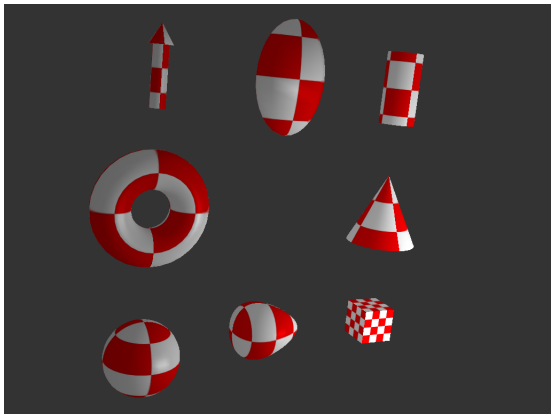

Figura 1.2: Principales primitivas en ISGL3D.

Para la creación de cada primitiva, se debe especificar además, la cantidad de segmentos utilizados en las distintas dimensiones. En la figura 1.2 se pueden ver todas las primitivas anteriores. Es fácil ver que dichas primitivas cuentan con cierta textura cuadriculada de colores rojo y blanco, que fue lograda mapeando una imagen sobre cada una de ellas:

```
Isgl3dTextureMaterial * material = [Isgl3dTextureMaterial
  materialWithTextureFile:@"red_checker.png" shininess:0.9];

Isgl3dTorus * torusMesh = [Isgl3dTorus meshWithGeometry:2 tubeRadius:1 ns:32 nt:32];

Isgl3dMeshNode * _torus = [self.scene createNodeWithMesh:torusMesh andMaterial:material];
```

En la primera línea de código se crea el material. Dicho material es del tipo *Isgl3dTextureMaterial*; y la imagen con la que este se creal es la de la figura 1.3. Luego, se crea el toroide asignándole los parámetros vistos más atrás en esta sección; y finalmente, se crea y se agrega a la escena el nodo asociado al toroide, con el material creado anteriormente.

A veces lo que se quiere no es agregar a la escena una primitiva sino un modelo previamente creado. Los modelos son realizados en herramientas de creado y animación de gráficos 3D como por ejemplo *Blender*, *MeshLab*, *Autodesk Maya* o *Autodesk 3ds Max*. Luego deben ser exportados en un formato llamado *COLLADA*, acrónimo de “COLLAborative Design Activity”, que sirve para el intercambio de contenido digital 3D entre distintas aplicaciones de modelado. Por su parte, ISGL3D permite importar modelos pero en un formato llamado *POD*. Se usó entonces, una aplicación llamada *Collada2POD* que lo que hace es convertir modelos tridimensionales en formato *COLLADA* al formato *POD*. *Collada2POD* puede ser descargada gratuitamente de la página oficial

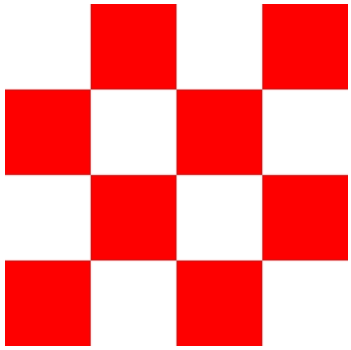

Figura 1.3: Imagen *red_checker.png*, utilizada para crear la textura asociada a las primitivas de la figura 1.2.

de *Imagination Technologies*, su desarrollador: `http://www.imgtec.com/`.

Una vez que se tiene al objeto 3D en el formato *POD*, este puede ser importado en ISGL3D de forma sencilla:

```
Isgl3dPODImporter * podImporter = [Isgl3dPODImporter podImporterWithFile:@‘‘modelo.pod’’];

Isgl3dNode * _model = [self.scene createNode];

[podImporter addMeshesToScene:_model];

 _model.position = iv3(2, 6, 0);
```

En la primera línea de código se instancia la clase *Isgl3dPODImporter* que sirve para transformar modelos POD a objetos ISGL3D, y se le asigna a la misma el modelo *modelo.pod*. Luego, se crea un nodo llamado “_model”, al que se le asignará el modelo; y se agrega a la escena. Finalmente, se le asigna al modelo una posición en la escena. En la figura 1.4 se puede ver un modelo de José Artigas, agregado dos veces a una misma escena, pero visto desde ángulos distintos.

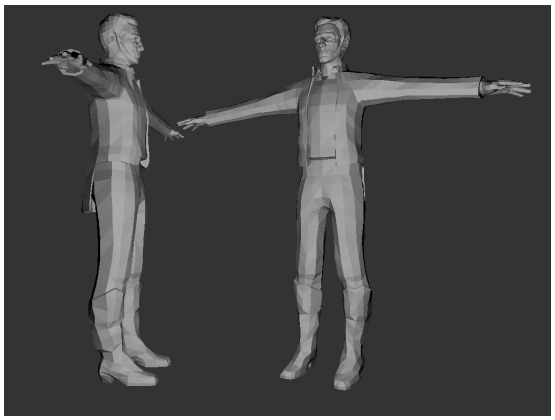

Figura 1.4: Dos vistas de un modelo de José Artigas en una misma escena.