Práctica de Visualización y Entornos Virtuales I

En la práctica comenzaremos a trabajar especialmente los siguientes temas:

- Geometría básica
- Transformaciones geométricas
- Grafo de la escena
- Cámara
- Avatar y colisiones

1 Geometría básica

- Implementar las siguientes funciones incluidas en el fichero `Math/line.cc`:
  - `void Line::setFromAtoB(const Vector3 & A, const Vector3 & B);`
  - `Vector3 Line::at(float u) const;`
  - `float Line::paramDistance(const Vector3 & P) const;`
  - `float Line::distance(const Vector3 & P) const;`
- Implementar las siguientes funciones incluidas en el fichero `Math/intersect.cc`:
  - `int BSpherePlaneIntersect(const BSphere *bs, Plane *pl);`
  - `int BBoxBBoxIntersect(const BBox *bba, const BBox *bbb );`
  - `int BBoxPlaneIntersect(const BBox *theBBox, Plane *thePlane);`

Nota: Para visualizar los resultados de este apartado utilizaremos el programa **Math/test**

2 Transformaciones geométricas

- Implementar las siguientes funciones incluidas en el fichero `Math/trfm3D.cc`:
  - `Vector3 Trfm3D::transformPoint(const Vector3 & P) const;`
  - `Vector3 Trfm3D::transformVector(const Vector3 & V) const;`
  - `void Trfm3D::setRotAxis(const Vector3 & V, const Vector3 & P, float angle );`

Nota: Para visualizar los resultados de este apartado utilizaremos el programa **browser_gobj** y realizaremos una modificación en el *shader* **Shader/dummy.vert**

3 Grafo de la escena

Vamos a realizar tres versiones y, por tanto, debemos entregar tres versiones del código (En nuestro caso, realizaremos tres *commits*):

1. Modo Local: Inicialmente podemos considerar que la transformación parcial de cada nodo gestionará la localización LOCAL (`m_placement`). Esto es, la posición y la orientación de un nodo viene dado por sus progenitores.
   - `addChild()`
   - `draw()`: Con transformación local
2. Modo global sin adaptación del BBox: En este caso se debe gestionar la transformación que localiza un nodo en el mundo directamente (`m_placementWC`). Esta última transformación actúa como una memoria caché y se utiliza para aligerar el cálculo computacional de la aplicación. Mediante ello, no se tienen que componer las transformaciones del nodo en cada *frame*. Eso sí, si movemos un nodo, debemos actualizar su transformación `m_placementWC` y también la de sus hijos!.
   - `updateWC()`
   - `updateGS()`: Llama a `updateWC()`
   - `addChild()`: Llama a `updateGS()`
   - `draw()`: Con transformación global
3. Modo global con adaptación del BBox
   - `updateBB()`
   - `propagateBBRoot()`
   - `updateGS()`: Llama a `updateWC()` y `propagateBBRoot()`

Nota: Para visualizar los resultados de este apartado utilizaremos el programa **browser**

4 Cámara

- Implementar las siguientes funciones incluidas en el fichero `Camera/camera.cc`:
  - `void PerspectiveCamera::updateProjection();`
  - `void Camera::lookAt(const Vector3 & E, const Vector3 & at, const Vector3 & up);`

5 Avatar y colisiones

- Implementar la siguiente función incluida en el fichero `Camera/avatar.cc` y comprobar que el *avatar* no se choca con los objetos de la escena.
  - `bool Avatar::advance(float step);`
- Por ello, vamos a implementar la siguiente función incluida en el fichero `Scene/node.cc`:
  - `const Node *Node::checkCollision(const BSphere *bsph) const;`