TP 5

Site: lms.univ-cotedazur.fr
Cours: Realite virtuelle - EIMAD919
Livre: TP 5

Imprimé par: Theo bonnet
Date: vendredi 28 février 2020, 14:56

Table des matières

1. Rebonds

Démarrage

Le point de départ du TP5 est le prolongement du Tuto5. Au début on continue à mettre en place des animations le long de trajectoires.

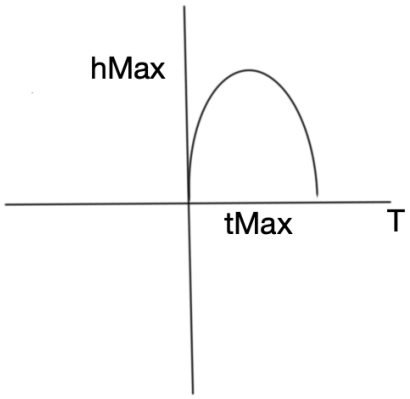
Rebonds

Créer une classe `RVBounce` qui hérite de `RVCurve` que l'on puisse utiliser comme trajectoire pur simuler le rebond de la sphère sur le plan : le but c'est que $y(t)$ soit une parabole renversée (coeff a négatif, alors que $x(t)$ et $z(t)$ soit constants.
Comme

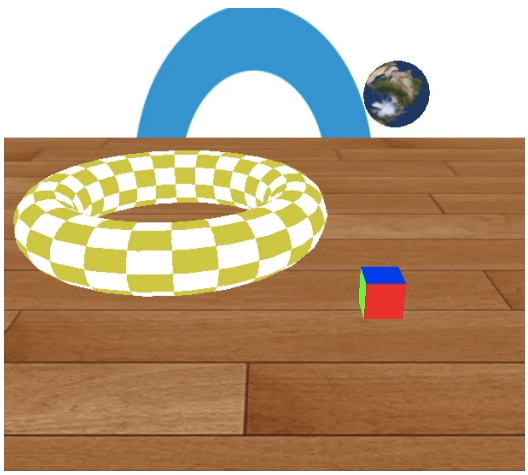
$$y(t) = at^2 + bt$$

avec $c = 0$ car on suppose que à l'instant $t = 0$ on a $y = 0$ les paramètres de `RVBounce` les réels a et b.
Mais le constructeur de la classe `RVBounce` foit permettre de choisir simplement la forme du rebond en donnant deux arguments :

- la hauteur maximale `hMax` à laquelle on arrive avant la descente,
- à quel moment on y arrive `tMax`



A vous de trouver les calculs permettant de trouver a et b en fonctions des deux paramètres et de montrer la balle rebondir sur le plan comme ci-dessous.



2. Vision Stéréoscopique

Présentation

La vision stéréoscopique c'est ce qui permet à notre cerveau d'avoir une reconstitution tridimensionnelle de notre environnement et donc d'évaluer les profondeurs : chaque oeil reçoit une image différente du monde et c'est en comparant ces deux images que l'on est capable de percevoir le relief.
Le cinéma 3D et la Réalité Virtuelle exploitent cette capacité du cerveau en offrant à chaque oeil une image légèrement différente du monde ; dans le premier cas avec des lunettes polarisées, dans le second cas grace à des casques spécialement conçus pour offri à chaque oeil une image différente.
Les premières expériences de cinéma 3D utilisaient le princpe des *anaglyphes* : on superpose les deux images, celle destinée à l'oeil gauche et celle destinée à l'oeil droit dans une même image en les multipliant respectivement par un filtre rouge et par un filtre cyan. Pour pouvoir regarder les anaglyphes il faut des lunettes aux verres teintés avec les bonnes couleurs.
Credits : je me suis beaucoup inspiré du document de Animesh Mishra « Rendering 3D Anaglyph in OpenGL » dont j'ai aussi reproduit les excellents diagrammes. A lire pour bien comprendre la notion de parallaxe.

Camera Stéréo

Camera Stéreo : ajouter au projet une classe `RVStereoCamera` qui hérite de `RVCamera` :

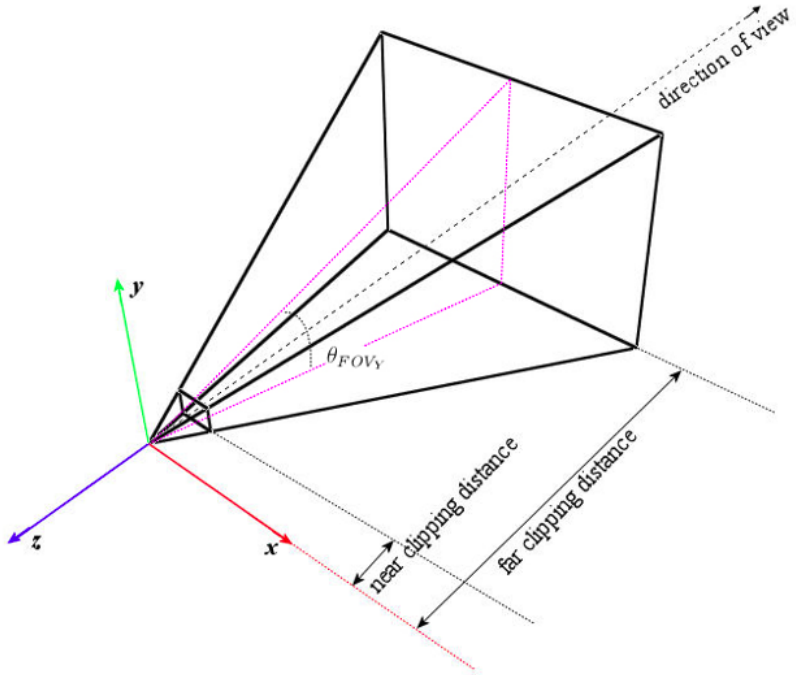
- Définir deux variables membres de type float pour définir la géométrie de la caméra stéréo : `m_eyeDistance` et `m_focalDistance` ; ajouter accesseurs et mutateurs.
- Définir une variable membre de type int appelée `m_cameraType` qui prendra pour valeur 0 pour la vue mono, 1 pour la vue gauche et 2 pour la vue droite ; le plus simple est de définir avec #define trois constantes RV_CAMERA_MONO, RV_CAMERA_LEFT, RV_CAMERA_RIGHT et les utiliser à la place des valeurs (plus élégant serait de définir une énumération...). Ajouter accesseur et mutateur.
- Dans le constructeur, appeler le constructeur de `RVCamera` puis initialiser la distance inter œil à 4 et la distance focale à 60 et le type de caméra à RV_CAMERA_MONO.
- Ajouter les méthodes

```
QMatrix4x4 leftViewMatrix();
QMatrix4x4 leftProjectionMatrix();
QMatrix4x4 rightViewMatrix();
QMatrix4x4 rightProjectionMatrix();
```

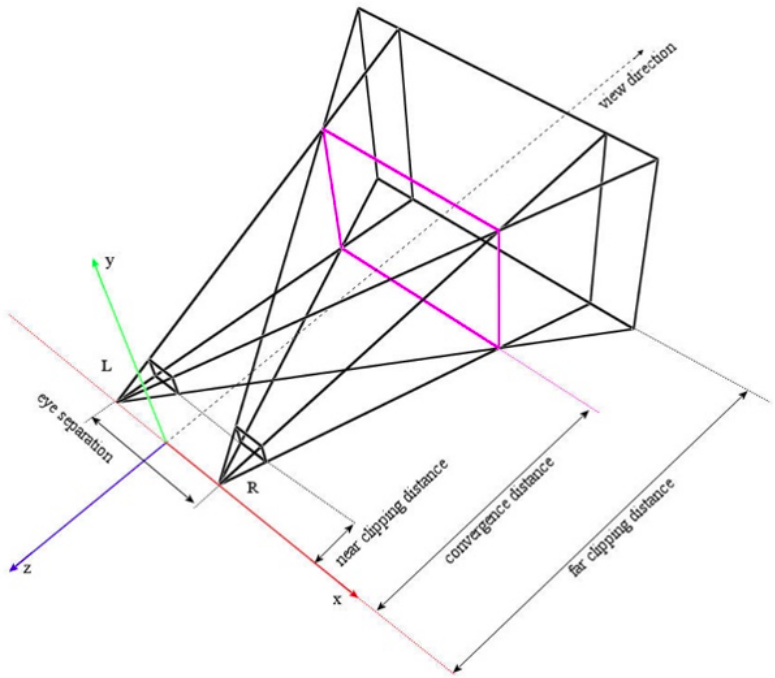
- Dans rvcamera.h rendre *virtual* les méthodes `viewMatrix()` et `projectionMatrix()` ; redéfinir ces méthodes dans RVStereoCamera.
- Implémentation de `RVStereoCamera::viewMatrix()` : selon la valeur de `m_cameraType` appeler soit `RVCamera::viewMatrix()`, soit `RVStereoCamera::leftViewMatrix()` soit `RVStereoCamera::rightViewMatrix()`. Même chose pour `RVStereoCamera::projectionMatrix()`.
- Implémentation de `leftViewMatrix()`.
  - Il s'agit de trouver la position de l'œil gauche et la cible correspondante ; pour cela il faut translater `m_position` et `m_target` sur la gauche selon une direction orthogonale à la fois à `m_up` et au vecteur de visée qui va de `m_position` à `m_target`, d'une quantité égale à la moitié de la distance entre les deux yeux. C'est à dire que si P est m_position et T est m_target on aura :

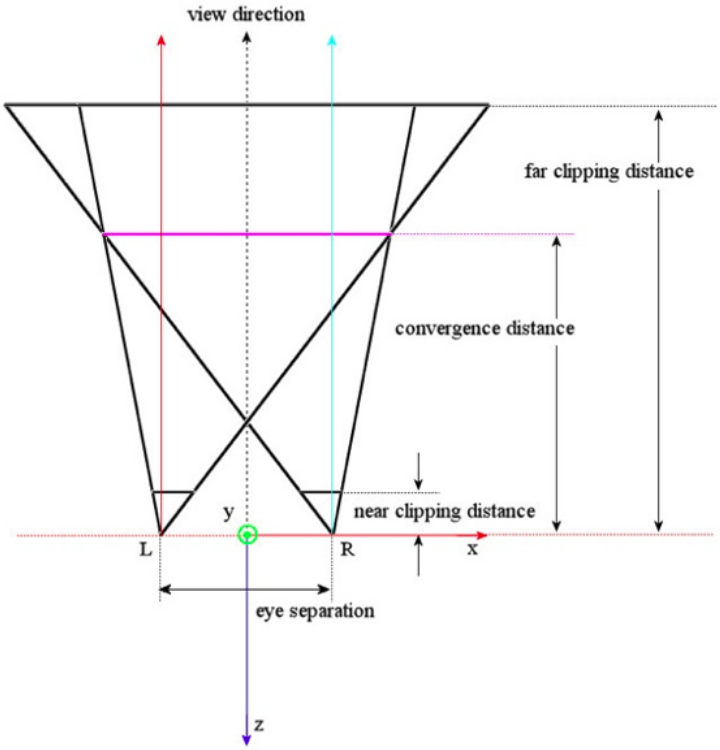
$$\begin{aligned} \vec{v} &= \overrightarrow{PT} \times \overrightarrow{up} \\ posOG &= P - \frac{d}{2}\frac{\vec{v}}{||v||} \\ cibleOG &= T - \frac{d}{2}\frac{\vec{v}}{||v||} \end{aligned}$$

    Avec posOG, cibleOG et m_up on peut définir la matrice de vue gauche avec la méthode lookAt de Matrix4x4.
  - Pour la vue droite c'est la même chose avec le signe positif.
- implémentation de `leftProjectionMatrix()` : on remplace la méthode `perspective()` utilisée dans RVCamera par `frustum()`. En effet perspective définit forcément un volume de vue de forme pyramidale (selon la direction des z négatifs) qui est symétrique à fois selon les plans horizontaux et verticaux passant par 0. Vous remarquez dans ce dessin que m_fov est l'ouverture verticale du vue.



- En revanche pour les matrices de projectiondes yeux gauche et droit, on ne peut plus utiliser la méthode `perspective` car les volumes de vue de chaque œil de la `RVStereoCamera` convergent sur le plan focal (situé à distance `m_focalDistance` de l'origine). Ils sont donc asymétriques.



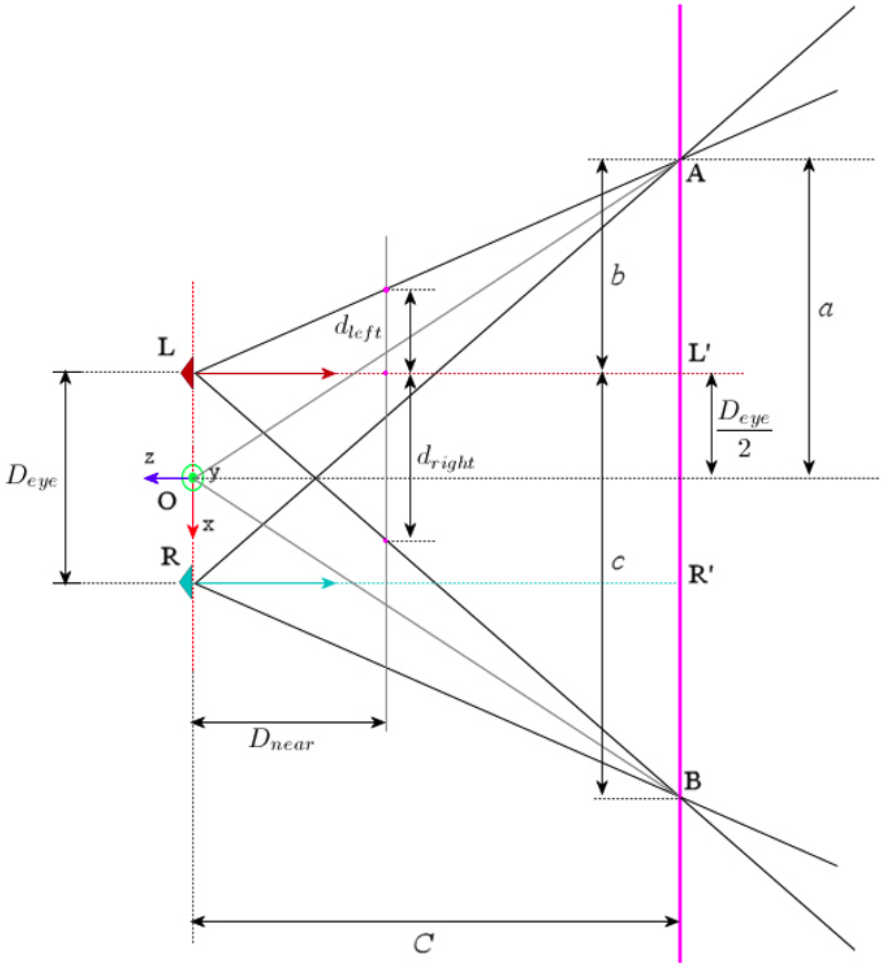


- On va donc utiliser la méthode `frustum` de `Matrix4x4` qui prend 6 arguments :

```
void QMatrix4x4::frustum(float left, float right, float bottom, float top,
float nearPlane, float farPlane)
```

  - Les 4 premiers arguments définissent les coordonnées du rectangle obtenu par l'intersection du volume de vue et du plan de fenêtrage avant (z = nearPlane = m_zMin)
  - Les 2 derniers sont les mêmes que m_zMin et m_zMax de RVCamera.

Alors il faut donc calculer les valeurs left, right, bottom, up à partir du dessin ci-dessous en connaissant `m_eyeDistance` (Deye), `m_focalDistance` (C) mais aussi à partir de `m_fov`, `m_aspect` et `m_zMin` (appelé Dnear dans le dessin) de RVCamera.



- Le calcul top et bottom est facile car

$$\tan(fov/2) = \frac{top}{zMin}$$

(attention à convertir l'angle en radians) donc

$$top = zMin \cdot \tan(fov/2)$$

et

$$bottom = -top.$$

Pour la camera mono (au centre), l'ouverture **horizontale** θ est telle que

$$\tan(\theta/2) = \frac{top \cdot aspect}{zMin}$$

et donc si on intersecte son volume de vue avec le plan focal (à distance C) on a :

$$\tan(\theta/2) = \frac{a}{C}.$$

Ce qui fait que l'on a :

$$a = \frac{C \cdot top}{zMin} \cdot aspect = C \cdot aspect \cdot \tan(fov/2).$$

Après on peut en déduire les valeurs de b et c (voir figure ci-dessus).

Pour trouver left et right (dleft et dright sur la figure) on utilise Thalès :

$$\frac{b}{C} = \frac{dleft}{zMin}$$

et

$$\frac{c}{C} = \frac{dright}{zMin}.$$

En fait dleft et dright sont des distances alors que left = -dleft et right = dright.

Utiliser la méthode frustum avec ces 6 valeurs pour renvoyer la matrice projection.

3. Utilisation de la caméra Stéréo

Utilisation de `RVStereoCamera` pour faire un anaglyphe :

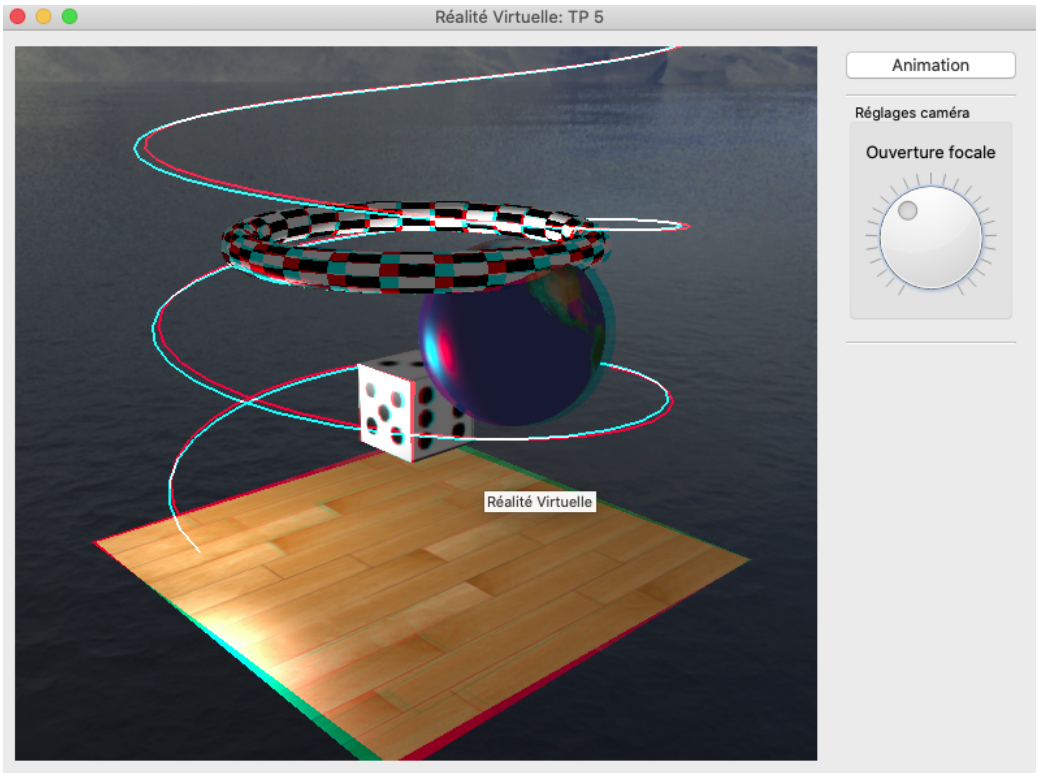
- Modifier `RVSphericalCamera` pour qu'elle hérite de `RVStereoCamera`
- Dans `paintGL()` de `RVWidget`
  - on utilise la vision MONO pour lerendu de la skybox
  - pour la vision de l'oeil gauche on utilise glColorMask pour mettre en place un filtre rouge

```
glColorMask(true, false, false, false);
```

  - on active dans `m_camera` le type RV_CAMERA_LEFT
  - on fait le rendu de la scène
- Pour le rendu œil droit :
  - On réinitialise le z-buffer

```
glClear(GL_DEPTH_BUFFER_BIT) ;
```

  - cette fois on active un filtre cyan
  - on choisit RV_CAMERA_RIGHT
  - on refait le rendu
- Enfin on remet le colorMask normal



4. Autres animations

Les prinicipes vus dans le tuto s'appliquent évidemment aussi à la caméra et à la lumière.

Vous pouvez par exemple placer la source lumineuse sur un cercle autour de la scène,

Concernant la caméra on peut fixer sa position sur une trajectoire mais on peut aussi fixer la cible sur une autre trajectoire. On a ainsi l'effet "visite guidée".

5. Effet 3D

Le but est d'accentuer l'effet de profondeur en créant beaucoup d'objets qui se déplacent dans la direction de l'observateur et lui passent derrière. En plus de l'effet de grossissement, cela va accentuer l'effet de décalage rouge/bleu qui est nul pour les objets dont la distance à l'observateur est égal à la distance focale.

Pour accentuer encore cet effet on utilise le principe des bandes noires (très utilisés dans les courts-métrages qui précèdent les films 3D au cinéma) : les bandes noires vont simuler des écrans de type cinéma. Mais en fait elles existent vraiment dans la scène (comme des RVPlan) à distance égale à la distance focale et on autorise les objets à "passer devant" pour avoir un effet de "sortie de l'écran".