Logiciels embarqués ambiants/iOS

Chapitre 3 : Rendu graphique avec OpenGL ES

Dr. Abdelkader Gouaïch[1]

[1]Department of Computer Science
Université de Montpellier

2012

Outline

Outline

Objetifs de ce chapitre

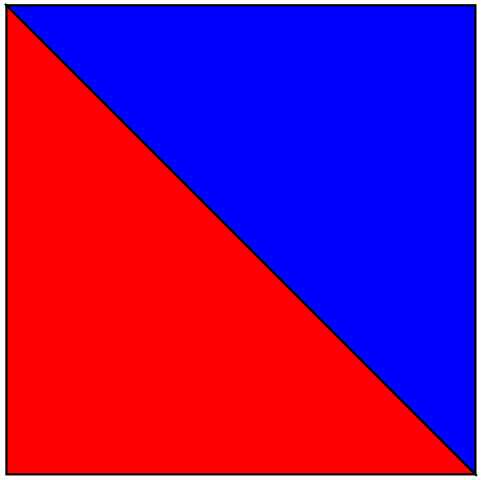
- Comprendre les bases d'OpenGL ES
- Comprendre le concept d'Interleaved Vertex Array
- Savoir dessiner une primitive géométrique avec OpenGL ES
- Comprendre le principe des textures
- Objective-C : comprendre les structures
- Comprendre les classes : Texture2D, TextureManager, ImageRenderManager, Image

- Nous allons aborder dans ce chapitre le traitement et le rendu des images en utilisant OpenGL ES
- Comment OpenCL ES peut nous rendre service afin de créer des jeux 2D iOS

Outline

Primitives graphiques dans OpenGL

- Open GL peut dessiner des triangles
- Toues les autres formes sont une combinaison de triangles
- Pour faire un rectangle par exemple il suffit de juxtaposer deux triangles



Principe général pour le moteur de jeu 2D

- Nous allons utiliser une astuce pour faire un jeu 2D en utilisant OpenGL (3D)
- Le principe général :
  - Nous allons construire des primitives géométriques : rectangles
  - Nous allons appliquer l'image source comme texture sur la primitive géométrique
- Optimisations :
  - Nous allons charger une image source (texture) qu'une seule fois
  - Une texture peut être utilisée plusieurs fois par plusieurs images
  - Nous cherchons à minimiser le nombre d'appels API OpenGL ES

Comment dessiner un carré sous open GL

- Pour dessiner un carré il faudrait dessiner deux triangles
- Nous allons utiliser un mode de dessin openGL qui est GL_TRIANGLES et GL_TRIANGLES_STRIP

Exemple d'utilisation

Example

```
glVertexPointer(2, GL_FLOAT, 0, squareVertices);
glColorPointer(4, GL_UNSIGNED_BYTE, 0, squareColors);
glDrawArrays(GL_TRIANGLE_STRIP,0,4);
```

- Nous avons utiliser ce code dans le TP pour déssiner un carré tableau
- paramètres : le mode de dessin, stride, nombre de points (vertices) à considérer

Le mode TRIANGLE_TRIP

- Nous allons faire un strip (enlever) des triangles
- OpenGL utilise toujours les deux derniers points + vertice pour créer un triangle
- Deux triangles successifs partagent deux vertices
- Exemple : P1 P2 P3 P4
- Deux triangles :
  - P1 P2 P3
  - P3 P2 P4
- Attention l'ordre des vertices est important !

Le mode GL_TRIANGLES

- Dans le mode GL_TRIANGLES nous devons présenter et dessiner les triangles indépendamment
- Nous allons présenter 3 vertices pour dessiner chaque triangle
- Pour faire un rectangle il nous faut donc 6 vertices

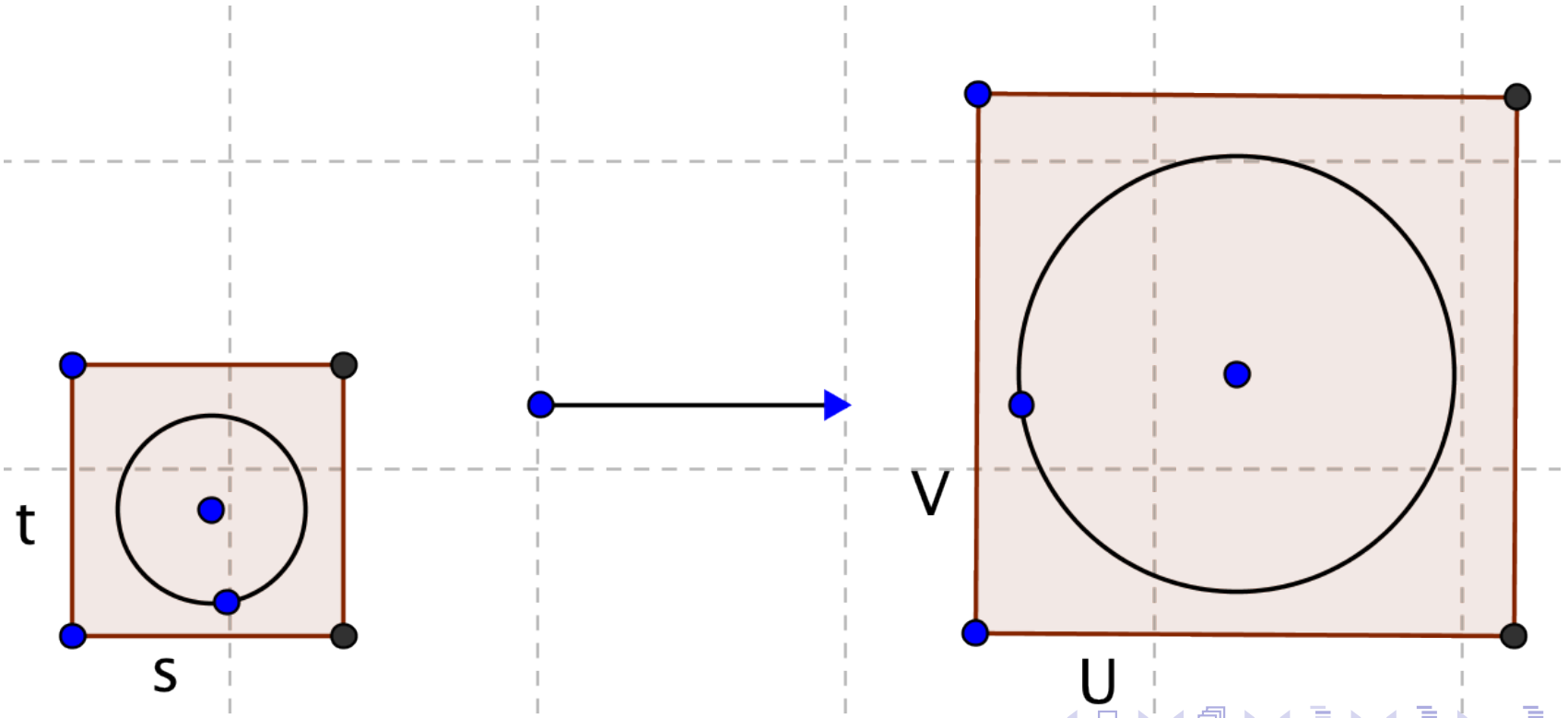
Outline

Principe de la texture

- C'est le processus de projection d'une texture (image) sur une primitive géo
- La project (mapping) c'est le lien entre les points de la primitive géométrique (x,y) et les points de l'image (s,t)

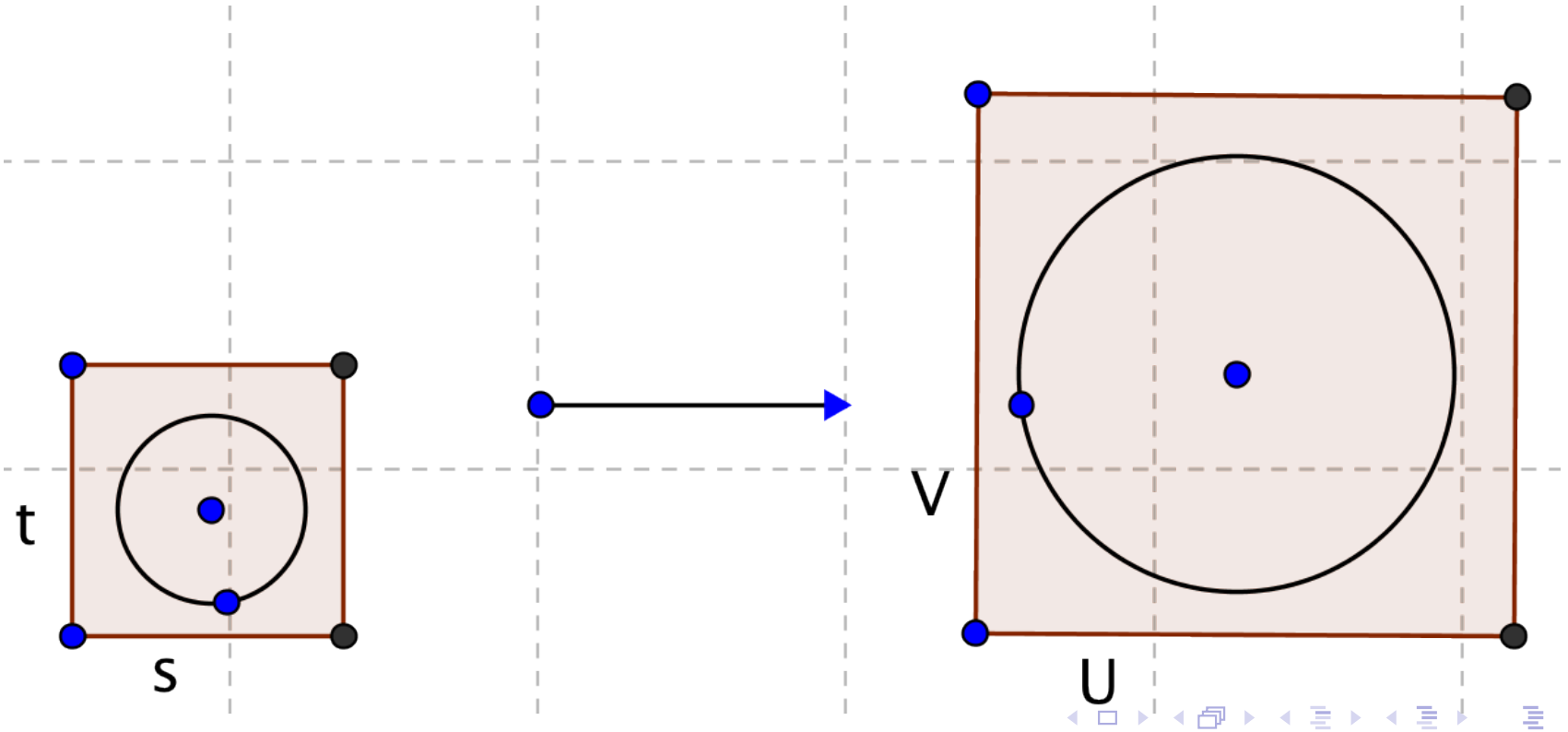
Coordonées de texture

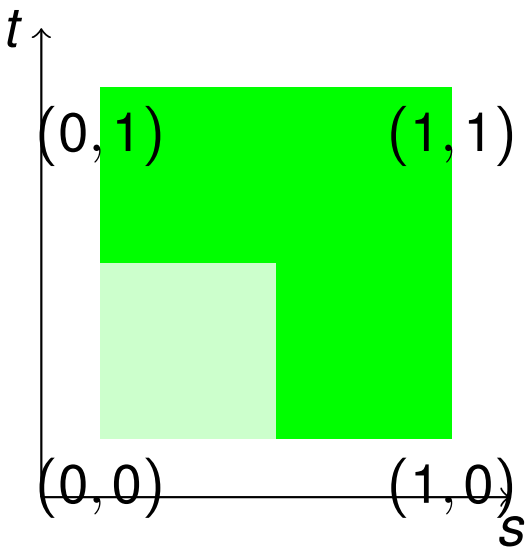
- Les coordonnées de texture sont s et t dans $[0, 1]$
- Les coordonnées de la primitive géo sont notées (u,v)
- UV mapping : Il s agit simplement du mapping de (s,t) dans (u,v)

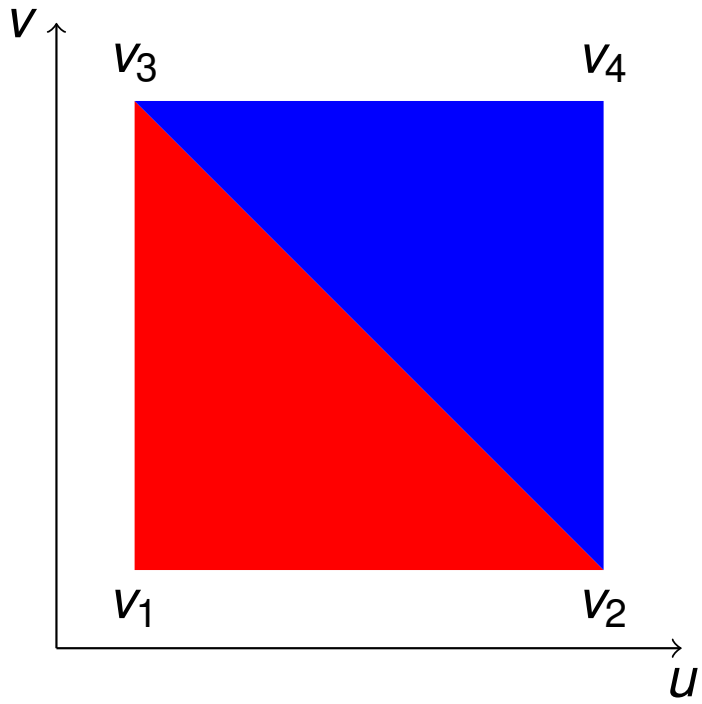


Coordonées de texture

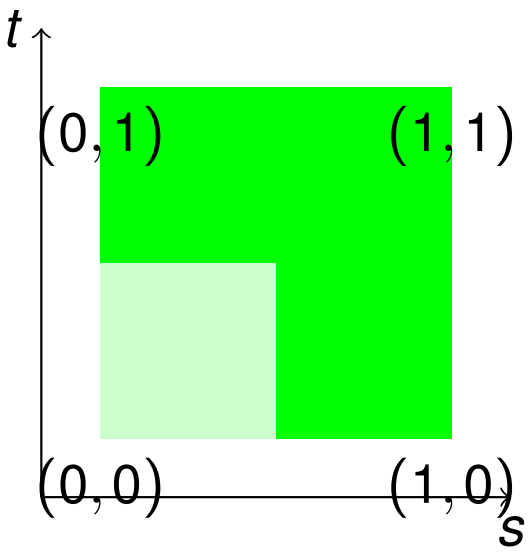
- Les coordonnées de texture sont s et t dans $[0, 1]$
- Les coordonnées de la primitive géo sont notées (u,v)
- UV mapping : Il s agit simplement du mapping de (s,t) dans (u,v)

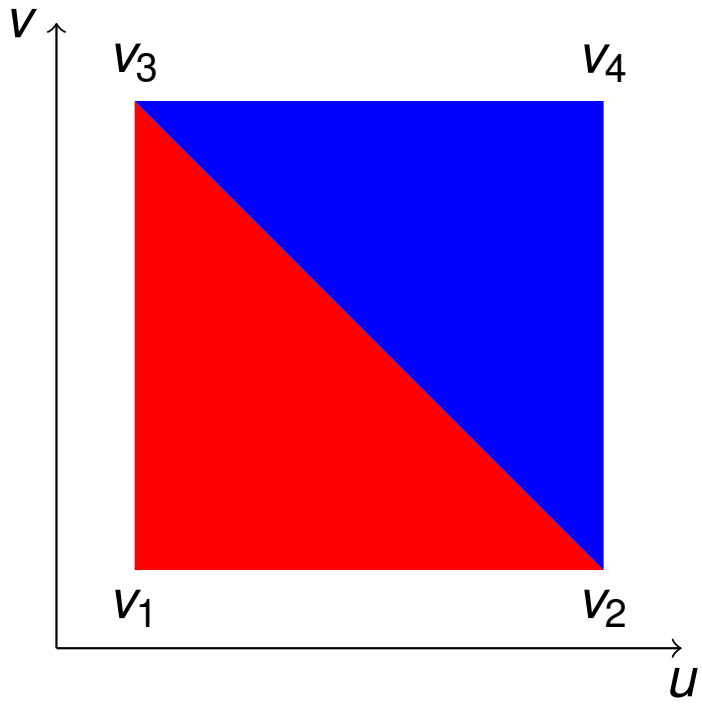






- mapping complet $(v_1, v_2, v_3, v_4) \mapsto (0,0), (1,0), (0,1), (1,1)$
- sous-image : $(v_1, v_2, v_3, v_4) \mapsto (0,0), (0.5,0), (0,0.5), (0.5,0.5)$





- mapping complet $(v_1, v_2, v_3, v_4) \mapsto (0,0), (1,0), (0,1), (1,1)$
- sous-image : $(v_1, v_2, v_3, v_4) \mapsto (0,0), (0.5,0), (0,0.5), (0.5,0.5)$

Outline

Présentation de l'objetif

- Nous avons utilisé deux tableaux (arrays) lors du TP :
  - un tableau pour ranger les points (vertices) de la primitive géo
  - un tableau pour ranger les couleurs des vertices
- Nous allons voir comment utiliser un seul tableau contenant les informations :
  - les coord. des vertices de la primitive géo
  - info sur la couleur
  - info sur la coordonnées de la texture

IVA

- Tableau vertex

x	y	x	y

- Tableau couleur

r	g	b	a	r	g	b	a

- Interleaver Vertex Array (IVA)

x	y	r	g	b	a	x	y	r	g	b	a

Définition du stride

- | x | y | r | g | b | a | x | y | r | g | b | a |
- Le stride donne l'espace (byte) entre deux points
- Stride vertex est de 24
- Stride color est de 24
- Pourquoi ? 4 byte * 6

Utilisation de l'IVA

Sans IVA

```
glVertexPointer(2, GL_FLOAT, 0, squareVertices)
;
glColorPointer(4, GL_FLOAT, 0, squareColors);
```

Avec IVA

```
glVertexPointer(2, GL_FLOAT, sizeof(GL_FLOAT)
*6, iva.geometry);
glColorPointer(4, GL_FLOAT, sizeof(GL_FLOAT)*6,
iva.color);
```

Structure dans objective-C

- Est un regroupement de variables
- Vous pouvez avoir des variables de différents types

TexturedColoredVertex

```
typedef struct {
CGPoint geometryVertex;
Color4f vertexColor;
CGPoint textureVertex;
} TexturedColoredVertex;
```

Exemple d'une structure pour ranger des informations :

- Coord. du point (géométrie)
- Couleur du point
- Coord. du mapping sur la texture

TexturedColoredVertex

```
typedef struct {
TexturedColoredVertex
        vertex1;
TexturedColoredVertex
        vertex2;
TexturedColoredVertex
        vertex3;
TexturedColoredVertex
        vertex4;
} TexturedColoredQuad;
```

Et pour une structure qui représente un rectangle

TexturedColoredVertex

```
typedef struct {
TexturedColoredQuad *
        texturedColoredQuad;
TexturedColoredQuad *
        texturedColoredQuadIVA
        ;
GLuint textureName;
} ImageDetails;
```

Structure pour avoir des informatins sur une image :

- les vertexes originaux de l'image (ne seront pas touchés par la suite)
- les vertexes utilisés qui pourront subir des transformations géo (translation,rotation, scale)
- l'identifiant (c'est un entier) de la texture chargée dans OpenGL

Outline

la classe Texture2D.m

- Son rôle est de créer une texture et la charger dans OpenGL à partir d'une image

```
@interface Texture2D : NSObject {
@private
GLuint name; // identifiant de la texture genere par opengl
CGSize contentSize; // la vraie taille de la texture (modifs.)
NSUInteger width; // vraies mesures de largeur et hauteur
NSUInteger height; //
GLfloat maxS; // En cas de modif de la taill avoir le max de s et t
GLfloat maxT;
CGSize textureRatio; // astuce de calcul pr rapidement la projection uv
Texture2DPixelFormat pixelFormat; // format de image source RGB, RGBA,A
}
```

la classse Texture2D.m

- Son rôle est de créer une texture et la charger dans OpenGL à partir d'une image
- une seule méthode :
  - `- (id) initWithImage:(UIImage*)aImage filter:(GLenum)aFilter;`
- aImage : l'image chargée (attention il y'a plusieurs formats)
- aFilter : filtre utilisé pour les opération de min et mag (réduction de la taille et augmentation de la taille)

initWithImage

Les étapes importantes pour cette méthode

1. Chargement de l'image source
2. Calcul de la nouvelle taille
3. Fabrication d'une nouvelle image avec la bonne taille et bon format de pixel
4. Demander à OpenGL de donner un nouveau nom de texture
5. Configuration OpenGL de la texture (min, mag)
6. chargement des données dans OpenGL
7. Calcul de maxS, maxT et textureRatio

Outline

TextureManager

- C'est un singleton qui va éviter de charger plusieurs fois la même texture
- Nous pouvons le voir donc comme un cache de texture

```
@interface TextureManager : NSObject {
    NSMutableDictionary *cachedTextures; // cache de
        textures
}
+ (TextureManager *)sharedTextureManager; //accesseur du
    singleton
// demande d'une texture avec un path de l'image source
- (Texture2D*)textureWithFileName:(NSString*)aName
    filter:(GLenum)aFilter;
//relacher une texture avec le path de l'image comme
    clef
- (BOOL)releaseTextureWithName:(NSString*)aName;
// relacher toutes les textures
- (void)releaseAllTextures;
@end
```

TextureManager

Exemple de la méthode textureWithFileName :

- regarder dans le cache si la clef existe déjà
- si retourner l'instance Texture2D existante
- sinon demander une nouvelle instance de Texture2D (initWithImage)
- mettre à jour le cache

Outline

ImageRenderManager

- Ce singleton va dessiner les images en utilisant OpenGL ES
- Important de bien comprendre cette classe pour comprendre OpenGL ES
- Attention ça va faire à la tête :)

```
@interface ImageRenderManager : NSObject {
    TexturedColoredVertex *iva; // tableau IVA pour vertexes à dessiner
    GLushort *ivaIndices;// tableau d'indices pour dire a OpenGL quels vertex
    //prendre dans l'IVA. Ceci est demande par la fonction glDrawElements
//indices des images dans l'IVA
// deux dimensions: [idTexture][#image]
    NSUInteger textureIndices[kMax_Textures][kMax_Images];
// pour un cycle R, la liste des textures a dessiner
// ie au moins une image est demandee
    NSUInteger texturesToRender[kMax_Textures];
    // nombre d'images pour cette texture
    NSUInteger imageCountForTexture[kMax_Textures];
    //nomber de texture
    NSUInteger renderTextureCount;
    // index courant dans l'IVA (nombre de quads)
    GLushort ivaIndex;
}
+ (ImageRenderManager *)sharedImageRenderManager;
// ajouter cette image dans la file de rendu
// en attendant le prochain cycle R
- (void)addImageDetailsToRenderQueue:(ImageDetails*)aImageDetails;
//ajouter les vertexes du quads dans l'IVA
- (void)addTexturedColoredQuadToRenderQueue:(TexturedColoredQuad*)aTCQ texture:(uint)aTexture;
//demande de dessin effective
- (void)renderImages;
@end
```

Outline

classe Image

- Pour le TP consultez Image.h et Image.m
- La méthode : initializeImage
- La méthode : render

Préparation du TP

- Téléchargez le fichier TP03.zip sur la page web
- Consultez dans le détail les classes étudiées dans le cours
- Consultez le fichier GameScene.m

Objetif du TP

1. Créer une application qui fait bouger l'image qui tourne et la faire rebondir quand elle arrive aux limites de l'écran
2. Tester l'application sur le simulateur
3. Option : Découvrez un bug dans l'application :)