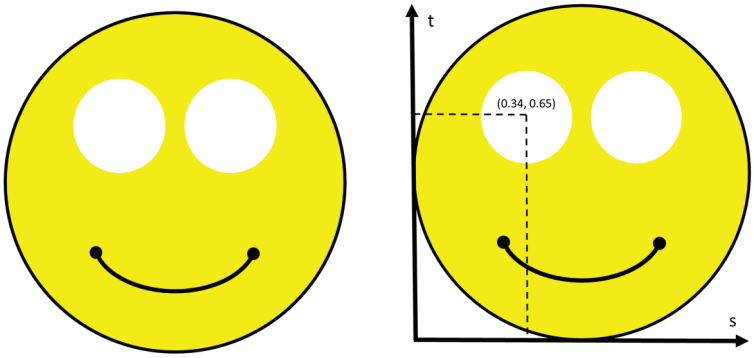
# Smile (smile.*) 2.5 punts

*Useu **/assig/grau-g/viewer** en tots els exercicis.*

Escriu **VS+FS** per texturar l'objecte **plane.obj** amb un *smile* amb ulls animats.

La textura que fareu servir (smile.png) té els ulls en blanc:

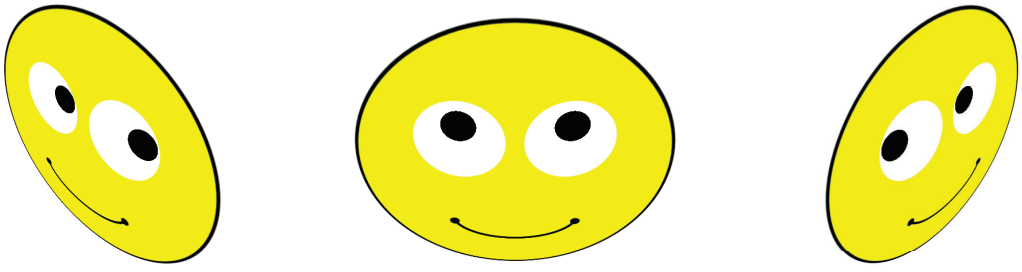



Els centres dels ulls tenen coordenades de textura C1= **(0.34, 0.65)** i C2=**(0.66, 0.65).**

El VS farà les tasques per defecte, però no ha de calcular cap color.

El FS bàsicament escriurà directament el color de la textura, amb l'excepció de l'iris de cada ull, que serà negre. Siguin (s,t) les coordenades de textura del fragment (vtexCoord). Si assumim que el centre de cada iris coincideix amb el centre de cada ull, el fragment haurà de ser negre si (s,t) és dins el cercle de radi 0.05 centrat en C1 o C2.

Volem però que l'iris es mogui com si mirés a la càmera:



Per aconseguir aquest efecte, considerarem que el centre de l'iris es desplaça respecte el centre de l'ull una distància que depèn de la normal N en *eye space*. Concretament, el desplaçament de C1 i C2 serà, per tots dos ulls, de **-0.1*N.xy**. Aquests nous centres són els que determinaran la posició dels iris.

**Identificadors obligatoris:**

smile.vert, smile.frag *(minúscules!)*
uniform sampler2D colormap;

# Ping Pong (pingpong.*) 5 punts

*Aquest exercici representa el 50% de la nota del control. Useu **/assig/grau-g/viewer***

Escriu **VS+GS+FS** per simular una pilota rígida que rebota en una taula (vegeu el vídeo **pingpong.ogv**).

Observeu que la pilota (sphere.obj) només es mou en direcció vertical (d'aquesta animació s'ocuparà el VS), i que estem dibuixant una taula i l'ombra de la pilota (això ho farà el GS).

El **VS** aplicarà a cada vèrtex una translació vertical a la seva coordenada Y en *object space*, d'acord amb l'equació del moviment uniformement accelerat:

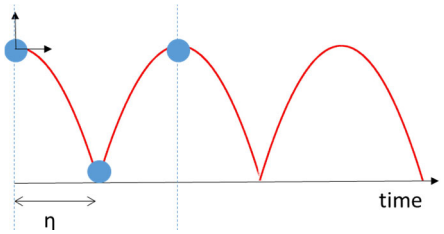
$$y(t) = y_0 + v_0 t + \tfrac{1}{2}at^2 \qquad (1)$$

on **$0 \le t < 2n$** és el temps modificat per a que l'animació es repeteixi periòdicament **cada 2n segons**, i n **és**

**uniform float n = 1;** // temps de caiguda lliure, en segons

L'acceleració (per la gravetat) serà constant: **const float a = -9.8.** Observeu el signe negatiu perquè la gravetat té la direcció de les Y negatives.

La posició inicial $y_0$ i la velocitat inicial $v_0$ dependran de t (la figura mostra com varia l'alçada del centre de la pilota en el temps):




Inicialment (t=0), la pilota està estàtica a l'origen i per tant la posició i velocitat inicials seran zero:

**$y_0 = 0$**
**$v_0 = 0.$**

Primera meitat del període (**t <= n**): la pilota cau per acció de la gravetat, amb els valors **$y_0 = v_0 = 0.$**

Al final de la primera meitat, la pilota impacta amb la taula**.** En aquest instant (**t=n)**, d'acord amb l'Equació (1), la posició de la pilota serà **$\frac{1}{2}an^2$** , i la seva velocitat serà **an**.

Segona meitat del període ( **n** < t < 2**n**): la pilota té trajectòria ascendent. Observeu que la posició i velocitats inicials per aquesta part seran, com hem vist:

**$y_0 = \frac{1}{2}an^2$**
**$v_0 = -an$** (considerarem que la velocitat simplement canvia de sentit, sense pèrdua d'energia).

Per tant, la translació (suma) que el VS aplicarà a la coordenada Y de cada vèrtex serà:

| | | |
|---|---|---|
| **$y_0 + v_0t + \frac{1}{2}at^2$** | amb **$y_0=0$, $v_0 = 0$** | si **$t < n$** |
| **$y_0 + v_0(t-n) + \frac{1}{2}a(t-n)^2$** | amb **$y_0 = \frac{1}{2}an^2$, $v_0 = -an$** | si **$n < t < 2n$** |

El VS li passarà al GS la posició gl_Position en **object space.** El color serà el color del VS per defecte (color multiplicat per la N.z en *eye space*).

El **GS** haurà d'emetre els triangles del model com habitualment (després de passar els vèrtexs a *clip space*). A més a més, **dibuixarà la taula i l'ombra** de la pilota.

Per tal d'evitar que el pla de retallat posterior retalli els objectes, el GS no farà servir directament EmitVertex(), sinó la següent funció, que escala les z (en *clip space*) per evitar aquest retallat:

```
void Emit()
{
  gl_Position.z /= 2.0;
  EmitVertex();
}
```

La taula és simplement un rectangle de **color gris R=G=B=0.5** situat al pla **y = ½an²-1** (restem 1 perquè l'esfera té aquest radi). Els vèrtexs de la taula són (en object space) de la forma **(± 5, ½an²-1, ± 5). El GS només pintarà la taula quan gl_PrimitiveIDIn sigui 0**.

Per l'ombra de la pilota, caldrà tornar a emetre el triangle inicial, però usant color negre i afegint com a transformació de modelat la matriu que projecta la pilota sobre la taula respecte un focus situat al punt (4,10,-4). Aquesta matriu la podeu obtenir amb la crida:

```
mat4 proj = projectOntoPlane();
```

El codi de la funció projectOntoPlane () el podeu trobar a **projectOntoPlane.glsl**.

El **FS** farà les tasques per defecte.

**Identificadors obligatoris:**

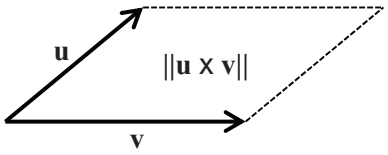
pingpong.* *(minúscules!)*
uniform float n = 1;
const float a = -9.8;

# Base (base.*) 2.5 punts

Escriu un **plugin** que escrigui pel canal de sortida estàndard (amb cout), l'àrea de la superfície de la meitat inferior del primer objecte de l'escena.

Només heu d'implementar **onPluginLoad**(). Aquest mètode recorrerà les cares del primer objecte de l'escena, i calcularà l'àrea de la seva superfície com la suma de l'àrea de les seves cares. Només cal sumar l'àrea de les cares on la seva y mínima sigui més petita que la Y del centre de la capsa englobant de l'objecte (feu servir el mètode **boundingBox()** de la classe Object).

Podeu assumir que els models seran malles de triangles. L'àrea d'un triangle es pot calcular fàcilment tenint en compte que el **mòdul del producte vectorial** de dos vectors **u** i **v** dóna l'àrea del paral·lelogram definit per **u** i **v**,




El mètode haurà d'escriure pel terminal l'àrea del primer objecte, precedida pel literal "Area:".

Aquí teniu un exemple de resultat, si només està carregat l'objecte per defecte:

Area: 18.6827

**Fitxers i identificadors (ús obligatori):**

`base.pro, base.h, base.cpp`