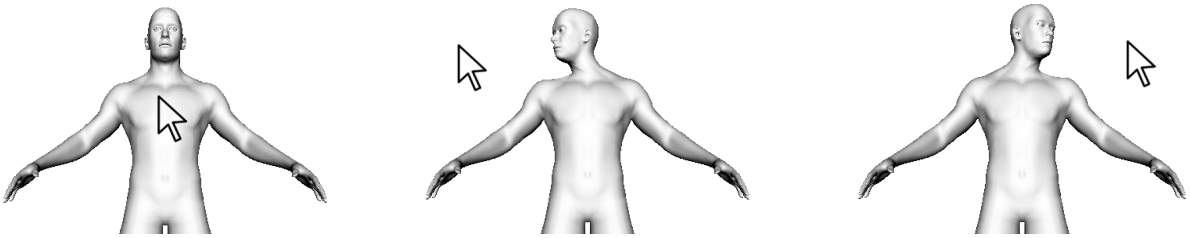
Look (look.*)

Escriu **VS+FS** per a girar el cap del model **man.obj** al voltant de l'eix Y, com si anés seguint el desplaçament horitzontal del cursor:



El VS farà les següents tasques. Primer, cal obtenir la coordenada X del cursor. Normalment usaríem

```
uniform vec2 mousePosition;
```

però per tal de fer possible el test, també usarem

```
uniform float mouseOverrideX = -1;
uniform vec2 viewport = vec2(800,600); // width & height
```

La coordenada X del cursor (en *window space*), serà `mousePosition.x` si `mouseOverrideX` és negatiu (valor per defecte); altrament serà `mouseOverrideX` (pels tests).

Assumirem que el cursor és dins la finestra del viewer i per tant que la coordenada X del cursor està en [0, viewport.x]. L'angle de rotació **alfa**, en radians, l'obtindrem transformant linealment la coordenada X del cursor a un valor entre [-1, 1]. D'aquesta forma, qual el cursor sigui a la vora esquerra, l'angle de rotació serà -1 rad, i quan sigui a la vora dreta l'angle serà 1 rad (aprox 57.3º).

Sigui P el vèrtex original, i sigui P' el resultat d'aplicar-li al vèrtex la rotació respecte l'eix Y (tots dos en *object space*). Volem que la rotació afecti només a la part superior del cos (coll i cap). Assumirem que el coll comença a Y=1.45 i acaba a Y=1.55.

La posició del nou vèrtex, encara en *object space*, l'heu de calcular com la interpolació lineal entre P i P' segons un paràmetre d'interpolació **t**. Aquest paràmetre ha de variar suaument (smootstep) entre 0 i 1, de forma que t=0 quan vertex.y<=1.45 (la rotació no tindrà efecte), t=1 quan vertex.y>=1.55 (rotació completa), i hi hagi una interpolació suau pels vèrtexs amb coordenada Y entre 1.45 i 1.55.

La nova normal l'heu de calcular interpolant la normal original amb la normal rotada, usant el mateix paràmetre d'interpolació t. Aquesta nova normal N, un cop transformada a *eye space*, és la que usareu per calcular el color del vèrtex, que tindrà per components RGB el valor de N.z.

Recordeu que la matriu de rotació respecte l'eix Y té la forma:

El FS farà les tasques per defecte.

$$\begin{bmatrix} cosa & 0 & sina & 0 \\ 0 & 1 & 0 & 0 \\ -sina & 0 & cosa & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Identificadors obligatoris:

look.vert, look.frag
Tots els uniform's de l'enunciat.

LightChange (lightChange.*)

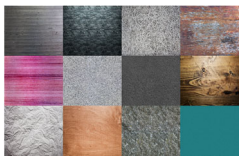
En aquest exercici us demanem **VS+FS** per simular una il·luminació basada en el model de Phong, amb diversos paràmetres que canvien en el temps.

El VS transformarà la normal i la posició del vèrtex a *eye space* i passarà aquestes dades al FS, juntament amb **la part fraccionària** de les coordenades de textura.

El FS calcularà la il·luminació usant el model de Phong, modificat com segueix:

- No hi haurà terme ambient.
- La llum difosa s'anirà encenent i apagant progressivament cada dos segons. El seu color (que substituirà a **lightDiffuse**) el calcularem com:
  - vec3(0) a t=0, 2, 4…
  - vec3(0.8) a t=1, 3, 5…
  - els gris que resulta d'interpolar linealment els valors anteriors, pels instants entremitjos (exemple, per t=2.5 el color serà vec3(0.4).
- La reflectivitat difosa del material (que substituirà a **matDiffuse**) s'obtindrà d'una textura amb múltiples imatges, on la imatge activa canviarà cada cop que la llum difosa s'apagui, seguint el següent ordre (per t entre 0 i 2 s'usarà la subimatge 1, i així successivament):

1	4	7	10
2	5	8	11
3	6	9	12



  Per prendre una mostra de la textura, useu les coordenades de textura del model (que estaran entre 0 i 1, ja que el VS n'haurà calculat la part fraccionària), convenientment transformades per referenciar la subimatge corresponent.
- Pel que fa al terme especular, calculeu-lo amb els uniforms habituals (matSpecular, lightSpecular, matShininess.
- Pels càlculs dels termes difós i especular, useu la posició de la llum **lightPosition**, com habitualment.

Aquí teniu el resultat esperat per t = 1.1, 3.1, 7.1, per diverses posicions de la llum:

Identificadors obligatoris:

lightChange.vert, lightChange.frag
uniform sampler2D colorMap;

Optics (optics.*)

En aquest exercici farem servir una escena fixa, formada per dos objectes: **sphere.obj** i **simpleplane.obj**.

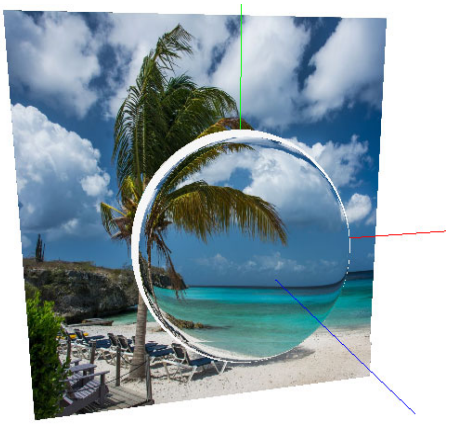
El primer és el model d'una esfera unitària centrada a l'origen, i el segon és el d'un quadrat que mesura quatre unitats de costat, centrat a l'eix z i **posicionat al pla z = -1.5**.

Voldrem simular com l'esfera, de cristall transparent, distorsiona la imatge que mostrem al quadrat. Per apreciar millor l'efecte, volem moure el pla allunyant-lo i acostant-lo en funció del valor de l'uniform float time. Concretament, en l'instant time, col·locarem el quadrat en

$$z = -1.5 - \sin^2(time)$$

Dins del FS, haurem de processar de forma diferent els fragments que provinguin del pla (que texturarem directament amb la textura al uniform sampler2D tex), i els fragments de l'esfera, per els quals simularem la refracció assumint un quocient d'índexs de refracció eta=1.7. En cap cas tindrem en compte la il·luminació.

Us recomanem que feu servir el test per a configurar l'escena i carregar una textura. Aquí teniu un exemple del resultat esperat:



Per implementar-lo disposes (a l'arxiu **trace.glsl**) de la funció

void trace(vec3 V, out vec3 P, out vec3 dir)

que donat un punt V a la superfície de l'esfera unitat, retorna el punt P per on un raig provinent de la càmera —incident a V— sortiria de l'esfera, i un vector unitari dir que indica en quina direcció ho faria (V, P, dir en *object space*). Si el raig (que conté punts de la forma P + λ dir, per un cert λ > 0) no interseca el quadrat, cal donar al fragment corresponent un color gris vec4(0.97). Altrament, cal donar-li el color que tingui el quadrat texturat en aquell punt.

Identificadors obligatoris:

uniform float time = 0;
uniform sampler2D tex;
const float eta = 1.7;