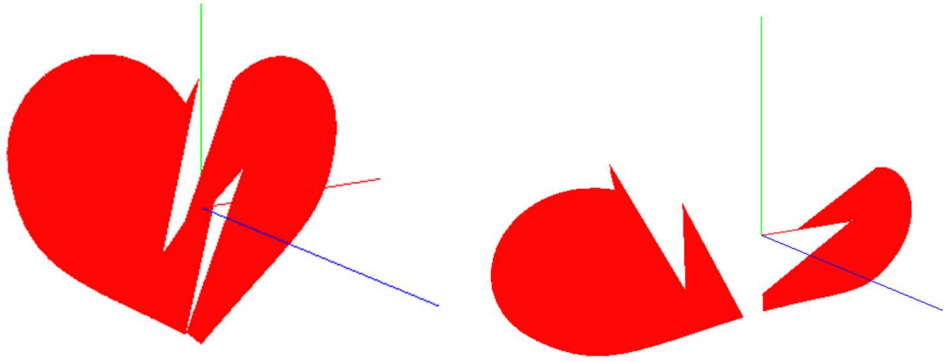
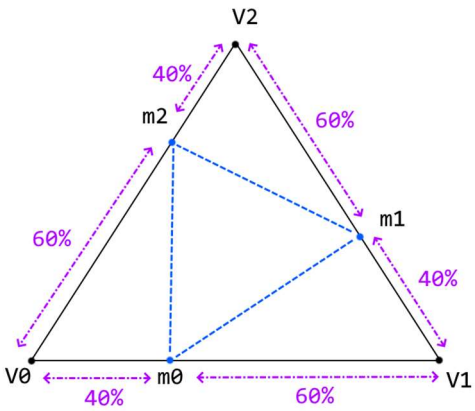
Broken Heart (brokenheart.*)

Escriu un **vertex shader**, un **geometry shader** i un **fragment shader** que generi un efecte de ruptura animada sobre un pla amb una textura aplicada. Podeu suposar llavors, que aquest shader el provarem només amb l'objecte Plane, que té coordenades de textura del (0,0) al (1,1).



1. El **vertex shader** farà les tasques imprescindibles.

2. El **geometry shader** haurà de subdividir cada triangle d'entrada en **4 triangles nous**. Per fer-ho, heu de **calcular nous vèrtexs sobre les arestes** del triangle d'entrada, fent per cada aresta una **interpolació lineal** dels seus dos vèrtexs amb **un pes fix de 0.4** (en tots els casos), i usar aquests punts per **construir els nous triangles**. **[4 punts]**



Un cop generats els nous triangles, heu d'animar-los i simular que tot es trenca separant-los en dues parts diferenciades, aplicant una rotació i una translació en espai objecte:

3. L'animació ha de ser **cíclica** amb un **període de 5 segons**. El **factor temporal `t`**, que controla la progressió de la separació i rotació, es calcula amb la fórmula:
`float t = smoothstep(0.0, 1.0, min(1.0, mod(time,5.0)/5.0))`
on `time` és el uniform que defineix el temps que ha passat en segons. **[1 punt]**

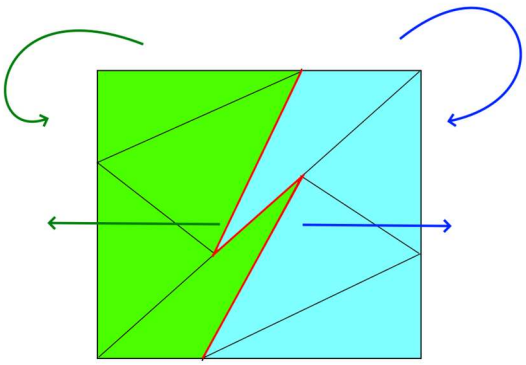
4. **Primer** s'ha de fer una **rotació en espai objecte** de **`t` radians al voltant de l'eix Z** que passa pel punt `(0,0,0)`. **[1 punt]**

Recordeu que la matriu de rotació sobre l'eix Z d'`α` radians és:

$$R_z(\alpha) = \begin{bmatrix} \cos(\alpha) & -\sin(\alpha) & 0 \\ \sin(\alpha) & \cos(\alpha) & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

5. **Després** de la rotació s'ha d'aplicar, de nou en **espai objecte**, **un desplaçament horitzontal sobre l'eix X** de magnitud `0.75 × t`. **[1 punt]**

6. Per **decidir cap a quin costat s'ha de rotar i moure cada triangle nou**, calculeu primer el **centre `C` del triangle subdividit abans d'aplicar qualsevol transformació**. Si cau a la part esquerra (`C.x < 0`), el triangle s'ha de moure i rotar cap a l'esquerra; si cau a la dreta (`C.x > 0`), ho farà cap a la dreta. **[1 punts]**



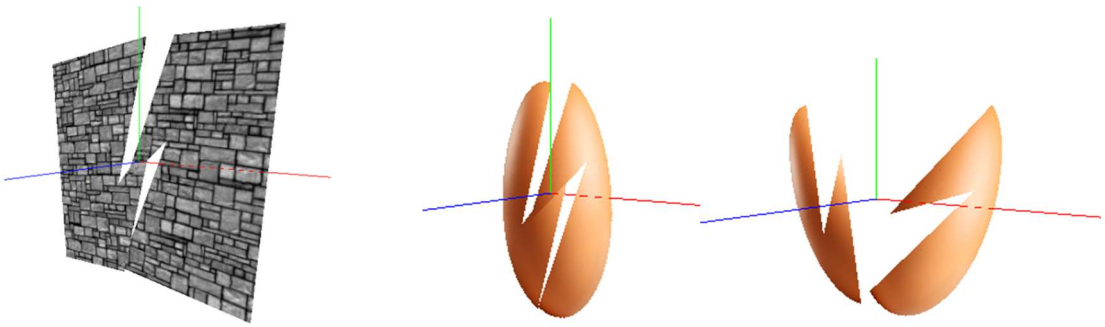
7. Recordeu que a més a més, **el geometry shader ha d'emetre les posicions dels vèrtexs en l'espai de coordenades adequat** perquè el pipeline d'OpenGL funcioni correctament. **[1 punt]**

8. El fragment shader ha d'aplicar una textura utilitzant les coordenades de textura que haurà de rebre del geometry shader, i descartar fragments amb alfa inferior a 1.0 per respectar les transparències de la textura. **[1 punt]**

Identificadors obligatoris:

`brokenheart.vert, brokenheart.geom, brokenheart.frag` (en minúscules!)

```
uniform float time;
```



Pokemon Card (pokemon-card.*)

En aquest exercici, volem implementar un efecte de brillantor hologràfica similar al de les cartes Pokemon amb "*foil*". Escriu un **Vertex Shader (VS)** i un **Fragment Shader (FS)** per aplicar aquest efecte a un quad bàsic (**plane.obj**), utilitzant diverses textures. El vídeo **pokemon-card.mp4** conté el resultat final esperat. L'exercici es puntua depenent de la variable **uniform int mode**; les puntuacions màximes que teniu per cada mode són orientatives.

		
mode = 0	**mode = 1**	**mode = 2**

<u>Si mode és 0</u> [4 punts]

El **VS** farà les tasques imprescindibles i ajustarà el quad perquè tingui una relació d'aspecte 3:4 (modifica només la coordenada X).

El **FS** s'encarregarà d'aplicar la textura **pikachu.jpg** a la part frontal i la textura **pikachu_back.jpg** a la posterior. Utilitza el vector **normal** del pla, i calcula el vector **V** (el mateix que fem servir a Phong) en **coordenades de món** per determinar si el fragment pertany a una cara frontal o posterior. La funció **dot(...)** et pot ser útil, però no és obligatori usar-la. Assegurat que la part posterior es veu correctament i no està invertida. Guarda el color final d'aquest mode en una variable amb nom **albedo**, que hauràs d'assignar al fragColor.

<u>Si mode és 1</u> [4 punts]

L'efecte d'aquest mode només s'aplica a la part frontal de la carta. Defineix la funció **vec3 foilColor(float hue)** al **FS** que converteix un valor de **hue** (entre 0 i 1) a un color RGB:

```
vec3 foilColor(float hue) {
    float saturation = 1.0;
    float value = 1.0;
    vec3 hsv = vec3(hue, saturation, value);
    // Convert HSV to RGB
    vec4 K = vec4(1.0, 2.0 / 3.0, 1.0 / 3.0, 3.0);
    vec3 p = abs(fract(hsv.xxx + K.xyz) * 6.0 - K.www);
    vec3 rgb = hsv.z * mix(K.xxx, clamp(p - K.xxx, 0.0, 1.0), hsv.y);
    return rgb;
}
```

Genera un valor de **hue** que canviï suaument i diagonalment segons la direcció de la vista. Per aconseguir-ho:

- Crea un vector **vec2 foilMotion** a partir de les components **x** i **y** del vector **V** calculat anteriorment (coordenades de món), i multiplicant-les pel vector **vec2(0.5, 0.3)**.
- Utilitza **foilMotion** per desplaçar les coordenades de textura **st** i obtenir **foilTexCoordOffset** (st + foilMotion).
- Calcula el **hue** com la part fraccionària de la suma de les components **x** i **y** del **foilTexCoordOffset**.

Usa el color **hue** i la funció **foilColor** per obtenir el color base de la brillantor (**foil**).

A continuació, usa el vector **V** (coordenades de món) per calcular el factor de reflexió per a que la brillantor sigui més intensa quan mirem la carta de costat. Per això, defineix la variable **reflectiveAngle**, que hauràs de calcular com el cosinus de l'angle entre el vector **V** i el vector **vec3(0.0, 0.0, 1.0)**. Finalment el factor de reflexió serà:

float reflectiveFactor = 1.4 - **reflectiveAngle**;
reflectiveFactor = pow(reflectiveFactor, 0.5);
reflectiveFactor = clamp(reflectiveFactor, 0.0, 1.0);

El color final serà:

fragColor = 1.0 - (1.0 - **albedo**) * (1.0 - vec4(**foil**, 1.0) * reflectiveFactor);

Si mode és 2 [2.0 punts]

El **FS** afegirà una textura de patró (**pikachu_foil.jpg**) a l'efecte foil del mode 1 (només a la part frontal de la carta). Per donar varietat al patró, mostreja la textura amb tres escalats diferents: **st**, **st * 0.5**, **st * 0.25**. Els tres colors resultants seran **p1**, **p2** i **p3**.

Després, per aconseguir l'efecte hologràfic, torna a utilitzar la funció **foilColor** per generar tres colors **f1**, **f2**, **f3**, a partir del **hue** calculat anteriorment. Per **f1** volem aplicar un *offset* al hue de 0.25, per **f2** de 0.5, i per **f3** de 0.75. Això farà que cada patró tingui colors diferents donat un mateix punt de vista. Compte que el **hue** ha d'estar en l'interval [0, 1].

Modifica la variable **foil** utilitzada anteriorment amb la següent expressió:

foil = foil * 0.7 + **f1** * **p1** + **f2** * **p2** + **f3** * **p3**;

Finalment, el **FS** aplicarà la màscara **pikachu_mask.jpg** a l'efecte hologràfic. Escala la variable **foil** per 0.75 on la màscara sigui blanca. Deixant la part negra sense escalar.

Identificadors obligatoris:

pokemon-card.vert, pokemon-card.frag (en minúscules!)
uniform int mode = 0;

La resta d'uniforms necessaris segons l'enunciat.

(*) Podeu lliurar els .vert, .frag de forma individual, o en un zip sense carpetes a dins.

drawinstances (drawinstances.*)

Useu GLarenaPL de la vostra instal·lació local del visualitzador

OpenGL ofereix la possibilitat de pintar moltes instàncies d'un mateix objecte, del que ja s'ha passat la informació del model a la GPU (creant el VAO i VBOs necessaris), de manera eficient. Per a això, disposa de la crida:

```
glDrawArraysInstanced (GL_TRIANGLES, 0, numVerts, n);
```

que permet enviar a pintar n instàncies del mateix model en la mateixa crida de pintat.

Per distingir aquestes instàncies i poder pintar-les de manera diferent, el Vèrtex Shader té una variable predefinida (int gl_InstanceID) que contindrà el valor entre 0 i n-1 que correspon a la instància que està pintant.

Us proporcionem l'esquelet d'un **plugin** (drawinstances.zip) que pinta l'escena amb uns shaders (també inclosos a l'esquelet) que fan la feina requerida per defecte i en el que s'indica (tant al plugin com als shaders), mitjançant el comentari // TODO: el que cal canviar per poder fer «*instancing*» i pintar diferents instàncies de l'objecte carregat.

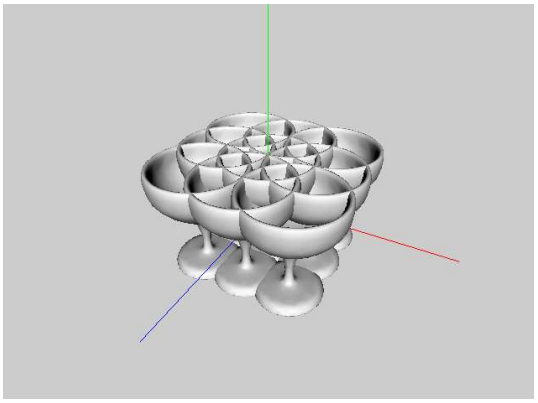
Concretament es demana:

1. Modifica el mètode **drawObject(…)** de l'esquelet de la manera adient per a què en lloc de pintar una única instància de l'objecte en pinti 9. **[2 punts]**

   Caldrà també que en el codi del Vèrtex Shader utilitzis la variable predefinida gl_InstanceID per fer un pintat de manera que els 9 objectes quedin en una graella de 3x3 en el pla XZ.

   En primer lloc, cal escalar tot l'objecte a la meitat de la seva mida **[1 punt]**. Tot seguit, cal disposar la graella de manera que els centres dels objectes adjacents, dins d'una mateixa fila o columna, estiguin separats per 1 unitat en coordenades de món. A més, els tres eixos de coordenades de món han de passar pel centre de la instància situada al mig de la graella **[3 punts]**.

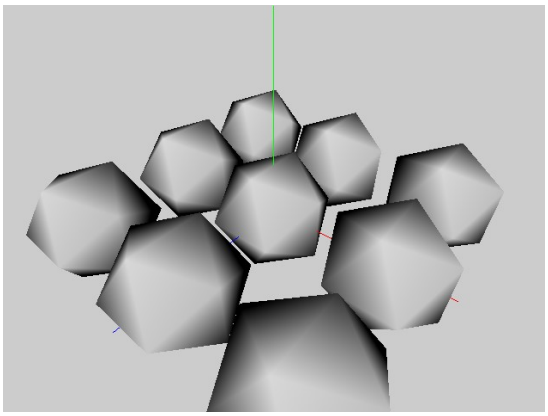
   Aquí teniu un exemple del resultat esperat per una escena amb default.obj

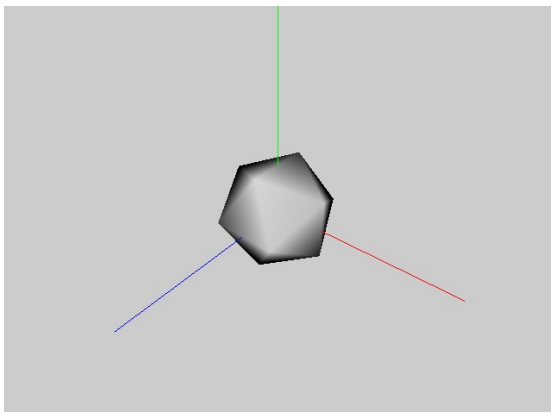


2. Afegeix ara la possibilitat que l'usuari, mitjançant les tecles `1` i `9` pugui decidir si pinta un únic objecte, escalat a la meitat però en la posició per defecte, o si en vol pintar 9 que li quedaran com hem descrit a l'apartat 1 **[3 punts]**.

   Per defecte s'han de veure les 9 instàncies **[1 punt]**.

   Aquí teniu un exemple del resultat esperat per una escena amb icosahedron.obj





Identificadors obligatoris:
drawinstances.cpp, drawinstances.h, drawinstances.pro instances.vert instances.frag
(en minúscules!)

Entregueu sols aquests 5 arxius dins una carpeta que es digui **drawinstances**, en un .zip, o .tar.