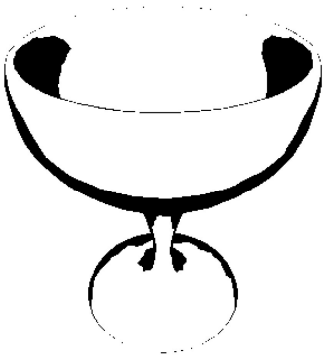
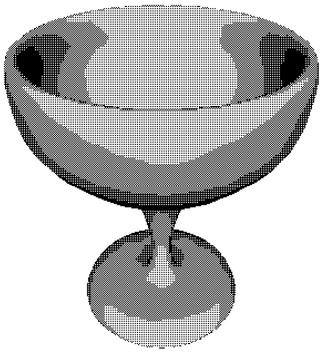
Dither (dither.*) Useu: `~/assig/grau-g/Viewer/GLarenaSL`

Volem visualitzar la malla fent servir només píxels de color blanc o de color negre. Per fer-ho podríem calcular una il·luminació bàsica de l'objecte. Els fragments que tinguessin una il·luminació major que 0.5 els podríem posar a blanc, i la resta a negre. Però si fem això, part de la forma de l'objecte es perdrà, com passa a la imatge de l'esquerra. Per millorar el resultat, aplicarem un efecte de dithering que ens donarà un resultat com el de la imatge de la dreta.





Escriu un VS+ FS per aconseguir aquest efecte. El **vertex shader** haurà de fer les tasques per defecte i calcularà la il·luminació fent servir la component z del vector normal en eye space.

El **fragment shader** haurà de:

Si el mode és 1, escriure negre si el valor d'il·luminació és menor que 0.5. Blanc en cas contrari.

Si el mode és 2, abans de decidir quin color escriure, modificarà el valor d'il·luminació sumant-li el següent:

- Si les coordenades x i y del fragment són parelles → -0.5
- Si la coordenada x és parella i la y senar → +0.25
- Si la coordenada x és senar i la y parella → +0
- Si totes dues coordenades són senars → -0.25

per finalment escriure blanc o negre segons el mateix llindar que amb el mode anterior. Quan comproveu la paritat de les coordenades x, y, considereu-ne únicament la part entera.

Identificadors obligatoris:

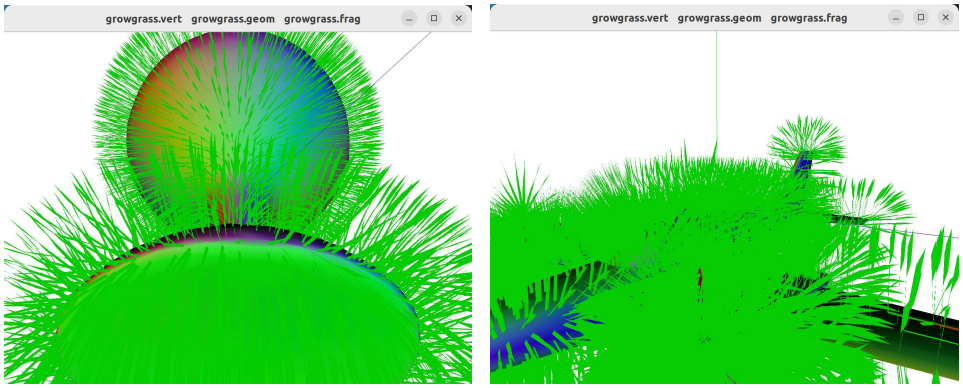
`dither.vert, dither.frag` (Has escrit **dither** correctament? En minúscules?)
`uniform int mode = 1;`

Grow grass (growgrass.*) Useu: `~/assig/grau-g/Viewer/GLarenaSL`

Escriu VS+GS+FS que dibuixi un bri de gespa al baricentre de cada triangle dels models.

Per tal de fer-ho, aprofitaràs la rutina `blade()` (a l'arxiu `blade.glsl`), que dibuixa un únic bri de gespa, en coordenades de model, que creix en la direcció de l'eix `Y`, i té la base a l'origen de coordenades. Hauràs de copiar el codi de la funció en el teu GS; observa que el codi suposa que tens un **`out`** `vec4 gfrontColor` ja declarat, i que és el que fa servir el FS per a assignar el color al fragment. De fet, no cal que modifiquis el FS per defecte, **però sí que cal** que l'incloguis en la teva entrega.

Vet aquí dos detalls del resultat amb els models `default.obj` i `cesna.obj`.



El teu VS haurà de produir en sortida les dades que necessiti el teu GS, en els sistemes de coordenades adequats.

El teu GS haurà de realitzar les següents tasques:

- Copiar en sortida cada primitiva, (amb els vèrtexs al sistema de coordenades escaient)
- Calcular la matriu de transformació que cal passar com argument a `blade()`. Aquesta matriu ha d'incloure diverses transformacions:
  - Escalar el bri de gespa de manera uniforme, amb un factor igual a la mida de diagonal de la capsa englobant del model dividida per 80,
  - Rotar-lo perquè tingui l'orientació de la normal de la cara (la mitjana de les normals dels tres vèrtexs del triangle), i traslladar-lo perquè la base sigui al baricentre del triangle (la mitjana de les coordenades dels tres vèrtexs),
  - Transformar de coordenades de model a les adequades a la sortida del GS.

Vigileu amb l'ordre en que multipliqueu les matrius per obtenir l'argument per `blade()`.

Per construir la matriu que rota i trasllada el bri de gespa a la seva posició final, tingues en compte que les seves columnes seran els vectors `cx, N, cz, C.`

$$\begin{bmatrix} cx.x & N.x & cz.x & C.x \\ cx.y & N.y & cz.y & C.y \\ cx.z & N.z & cz.z & C.z \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

- `N` és la normal unitària de la cara en coordenades del model,
- `C` és el vector de posició del baricentre de la cara, i
- `cx` i `cz` són vectors unitaris coincidents amb els vectors unitaris en la direcció x i la direcció z quan la normal és (0, 0, ±1). Altrament calcularem `cx` normalitzant el producte vectorial de `N` amb (0,0,1), i calcularem `cz` normalitzant el producte vectorial de `cx` amb `N`.

Identificadors obligatoris:

`growgrass.vert, growgrass.geom, growgrass.frag` **(en minúscules!)**

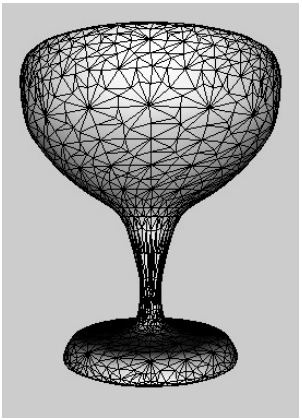
La resta d'uniforms estàndard necessaris segons l'enunciat: `uniform vec3 boundingBoxMin`...

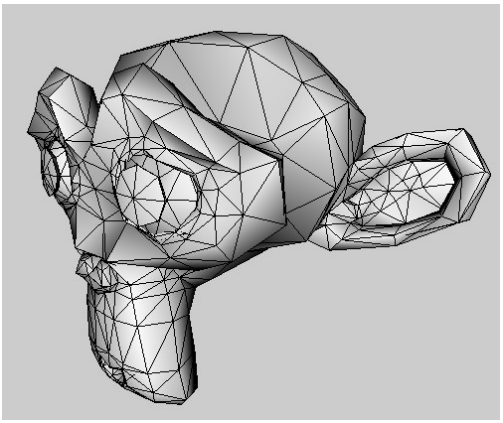
Gràfics
Curs 2022/23 Q2

Wire (Wire.*)

Useu `GLarenaPL` de la vostra instal·lació local del visualitzador

Escriu un **render plugin** que, utilitzant dues passades de rendering, permeti visualitzar el model amb els triangles omplerts de color, juntament amb el seu contorn de color negre:





El plugin caldrà que implementi les funcions

```
void onPluginLoad();
bool paintGL();
```

En el primer pas de renderització **(6 punts)**, caldrà pintar l'escena amb VS+FS que apliquin il·luminació bàsica: el color final del fragment tindrà per components RGB la component Z de la normal en *eye space*. El codi dels shaders ha d'estar al mateix fitxer cpp (com a l'exemple effectCRT).

En el segon pas de renderització **(3 punts)**, caldrà tornar pintar l'escena, amb els mateixos shaders, però enviant-li un uniform addicional per tal que el color del fragment sigui negre. Per tal de dibuixar només el contorn dels triangles, haureu d'usar **glPolygonMode** (més avall en teniu la descripció).

Feu servir glPolygonOffset **(1 punt)** per tal de donar prioritat de visibilitat als contorns negres dels triangles, per evitar que quedin parcialment ocultats pels fragments del primer pas.

Identificadors obligatoris:

`Wire.cpp, Wire.h, Wire.pro` (segur que has escrit **Wire** correctament?)

glPolygonMode *— select a polygon rasterization mode*

void glPolygonMode(GLenum face, GLenum mode);

face - Specifies the polygons that mode applies to. Must be GL_FRONT_AND_BACK for front- and back-facing polygons.

mode - Specifies how polygons will be rasterized. Accepted values are GL_LINE, and GL_FILL. GL_LINE - Boundary edges of the polygon are drawn as line segments. GL_FILL - The interior of the polygon is filled.

glPolygonOffset *— set the scale and units used to calculate depth values*

void glPolygonOffset(GLfloat factor, GLfloat units);

factor - Specifies a scale factor that is used to create a variable depth offset for each polygon. The initial value is 0.

Units - Is multiplied by an implementation-specific value to create a constant depth offset. The initial value is 0.

When GL_POLYGON_OFFSET_FILL or GL_POLYGON_OFFSET_LINE is enabled, each fragment's depth value will be offset after it is interpolated from the depth values of the appropriate vertices. The value of the offset is factor×DZ+r×units, where DZ is a measurement of the change in depth relative to the screen area of the polygon, and r is the smallest value that is guaranteed to produce a resolvable offset for a given implementation. The offset is added before the depth test is performed and before the value is written into the depth buffer.