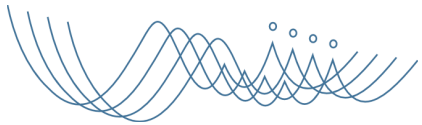
El Codi dels Shaders



Algunes diferencies respecte ShaderMaker

Versions

- Al laboratori teniu disponible la versió 3.3 (330)
- La primera línia que no està en blanc ni és comentari dels vostres shaders hauria de ser

  ```
  #version 330 compatibility
  ```

- Podeu usar versions inferiors, si voleu, però no podeu barrejar en un mateix programa un shader de la versió 330 amb un de versió 130 o menor.
- Si no hi ha declaració de versió, s'assumeix 110
- Podeu fer servir el `core` *profile* posant

  ```
  #version 330 core
  ```

  (o no posant cap *profile*), però aleshores haureu de gestionar vosaltres les matrius de transformació necessàries i passar-les com `uniforms`.

Algunes diferencies respecte ShaderMaker

Versions

- Al laboratori teniu disponible la versió 3.3 (330)
- La primera línia que no està en blanc ni és comentari dels vostres shaders hauria de ser

  ```
  #version 330 compatibility
  ```

- Podeu usar versions inferiors, si voleu, però no podeu barrejar en un mateix programa un shader de la versió 330 amb un de versió 130 o menor.
- Si no hi ha declaració de versió, s'assumeix 110
- Podeu fer servir el `core` *profile* posant

  ```
  #version 330 core
  ```

  (o no posant cap *profile*), però aleshores haureu de gestionar vosaltres les matrius de transformació necessàries i passar-les com `uniforms`.

Algunes diferencies respecte ShaderMaker

Versions

- Al laboratori teniu disponible la versió 3.3 (330)
- La primera línia que no està en blanc ni és comentari dels vostres shaders hauria de ser

  ```
  #version 330 compatibility
  ```

- Podeu usar versions inferiors, si voleu, però no podeu barrejar en un mateix programa un shader de la versió 330 amb un de versió 130 o menor.
- Si no hi ha declaració de versió, s'assumeix 110
- Podeu fer servir el `core` *profile* posant

  ```
  #version 330 core
  ```

  (o no posant cap *profile*), però aleshores haureu de gestionar vosaltres les matrius de transformació necessàries i passar-les com `uniforms`.

Algunes diferencies respecte ShaderMaker

Versions

- Al laboratori teniu disponible la versió 3.3 (330)
- La primera línia que no està en blanc ni és comentari dels vostres shaders hauria de ser

  ```
  #version 330 compatibility
  ```

- Podeu usar versions inferiors, si voleu, però no podeu barrejar en un mateix programa un shader de la versió 330 amb un de versió 130 o menor.
- Si no hi ha declaració de versió, s'assumeix 110
- Podeu fer servir el `core` *profile* posant

  ```
  #version 330 core
  ```

  (o no posant cap *profile*), però aleshores haureu de gestionar vosaltres les matrius de transformació necessàries i passar-les com `uniforms`.

Algunes diferencies respecte ShaderMaker

Versions

- Al laboratori teniu disponible la versió 3.3 (330)
- La primera línia que no està en blanc ni és comentari dels vostres shaders hauria de ser

  ```
  #version 330 compatibility
  ```

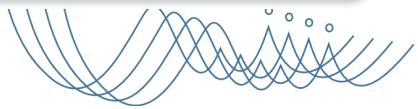
- Podeu usar versions inferiors, si voleu, però no podeu barrejar en un mateix programa un shader de la versió 330 amb un de versió 130 o menor.
- Si no hi ha declaració de versió, s'assumeix 110
- Podeu fer servir el `core` *profile* posant

  ```
  #version 330 core
  ```

  (o no posant cap *profile*), però aleshores haureu de gestionar vosaltres les matrius de transformació necessàries i passar-les com `uniforms`.

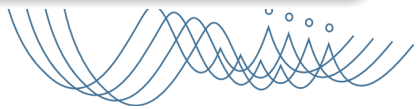
Algunes diferències sintàctiques

- `attribute` i `varying` estàn "deprecated"; ara les entrades d'un shader es declaren com `in` i les sortides com a `out`.
- Ara disposem de *layout qualifyers* que poden fer-se servir per especificar múltiples aspectes de la interfície entre shaders i el seu entorn. Farem servir:
  ```
  layout (triangles) in;
  layout (triangle_strip, max_vertices=6) out;
  ```
  per a especificar les entrades i sortides dels *geometry shaders*. Naturalment això és un exemple, i si s'escau farem servir altres primitives o nombres de vèrtexs...
- Podeu controlar la interpolació dels valors que es passen de vèrtexs a fragments amb els modificadors `smooth` (que és el valor per defecte), `flat` (l'anul·la; el `provoking vertex` s'imposa) i `noperspective` (fa interpolació lineal, sense correcció perspectiva).



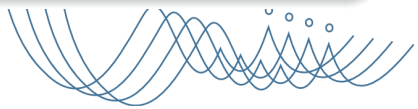
Algunes diferències sintàctiques

- `attribute` i `varying` estàn "deprecated"; ara les entrades d'un shader es declaren com `in` i les sortides com a `out`.
- Ara disposem de *layout qualifyers* que poden fer-se servir per especificar múltiples aspectes de la interfície entre shaders i el seu entorn. Farem servir:
  ```
  layout (triangles) in;
  layout (triangle_strip, max_vertices=6) out;
  ```
  per a especificar les entrades i sortides dels *geometry shaders*. Naturalment això és un exemple, i si s'escau farem servir altres primitives o nombres de vèrtexs...
- Podeu controlar la interpolació dels valors que es passen de vèrtexs a fragments amb els modificadors `smooth` (que és el valor per defecte), `flat` (l'anul·la; el `provoking vertex` s'imposa) i `noperspective` (fa interpolació lineal, sense correcció perspectiva).



Algunes diferències sintàctiques

- `attribute` i `varying` estàn "deprecated"; ara les entrades d'un shader es declaren com `in` i les sortides com a `out`.
- Ara disposem de *layout qualifyers* que poden fer-se servir per especificar múltiples aspectes de la interfície entre shaders i el seu entorn. Farem servir:
  ```
  layout (triangles) in;
  layout (triangle_strip, max_vertices=6) out;
  ```
  per a especificar les entrades i sortides dels *geometry shaders*. Naturalment això és un exemple, i si s'escau farem servir altres primitives o nombres de vèrtexs...
- Podeu controlar la interpolació dels valors que es passen de vèrtexs a fragments amb els modificadors `smooth` (que és el valor per defecte), `flat` (l'anul·la; el `provoking vertex` s'imposa) i `noperspective` (fa interpolació lineal, sense correcció perspectiva).

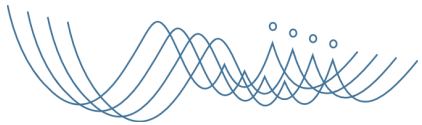


Algunes diferències sintàctiques (II)

- Els noms de les funcions d'accés a textures han estat simplificats. Allà on posàvem `texture2D(...)` ara posarem simplement `texture(...)`.
- En comptes de llegir uns arrays amb els noms dels diferents atributs canviats, els geometry shaders ara tenen per entrada un array de structs `gl_in`:

```
1 in gl_PerVertex{
2     vec4 gl_Position;
3     float gl_PointSize;
4     float gl_ClipDistance[];
5 } gl_in[];
```

  Podem consultar la seva mida amb `gl_in.length()`



Algunes diferències sintàctiques (II)

- Els noms de les funcions d'accés a textures han estat simplificats. Allà on posàvem `texture2D(...)` ara posarem simplement `texture(...)`.
- En comptes de llegir uns arrays amb els noms dels diferents atributs canviats, els geometry shaders ara tenen per entrada un array de structs `gl_in`:

```
1 in gl_PerVertex{
2    vec4 gl_Position;
3    float gl_PointSize;
4    float gl_ClipDistance[];
5 } gl_in[];
```

Podem consultar la seva mida amb `gl_in.length()`