Transparence & Geometry Shader

Présentation

David Odin

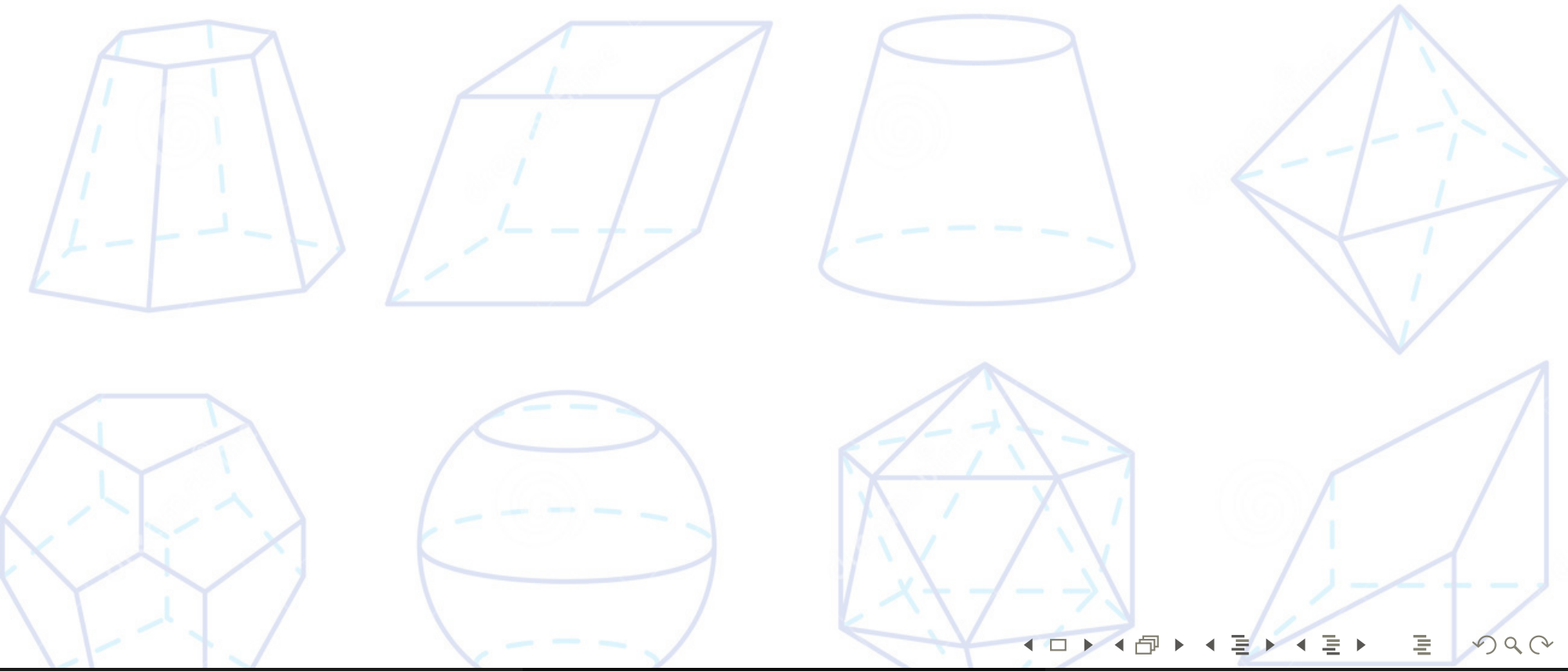
Forma3Dev pour CPE–Lyon

2016

Transparence et geometry shader – Plan

Un nouvel étage

- Après le Vertex Shader



Un nouvel étage

- Après le Vertex Shader
- Avant la rasterization (le découpage en fragments/pixels)

Un nouvel étage

- Après le Vertex Shader
- Avant la rasterization (le découpage en fragments/pixels)
- Avant un éventuel TransformFeedback

Un nouvel étage

- Après le Vertex Shader
- Avant la rasterization (le découpage en fragments/pixels)
- Avant un éventuel TransformFeedback
- Permet de changer, de supprimer, ou d'ajouter des primitives.

Un nouvel étage

- Après le Vertex Shader
- Avant la rasterization (le découpage en fragments/pixels)
- Avant un éventuel TransformFeedback
- Permet de changer, de supprimer, ou d'ajouter des primitives.
- Peut utiliser des variables uniformes

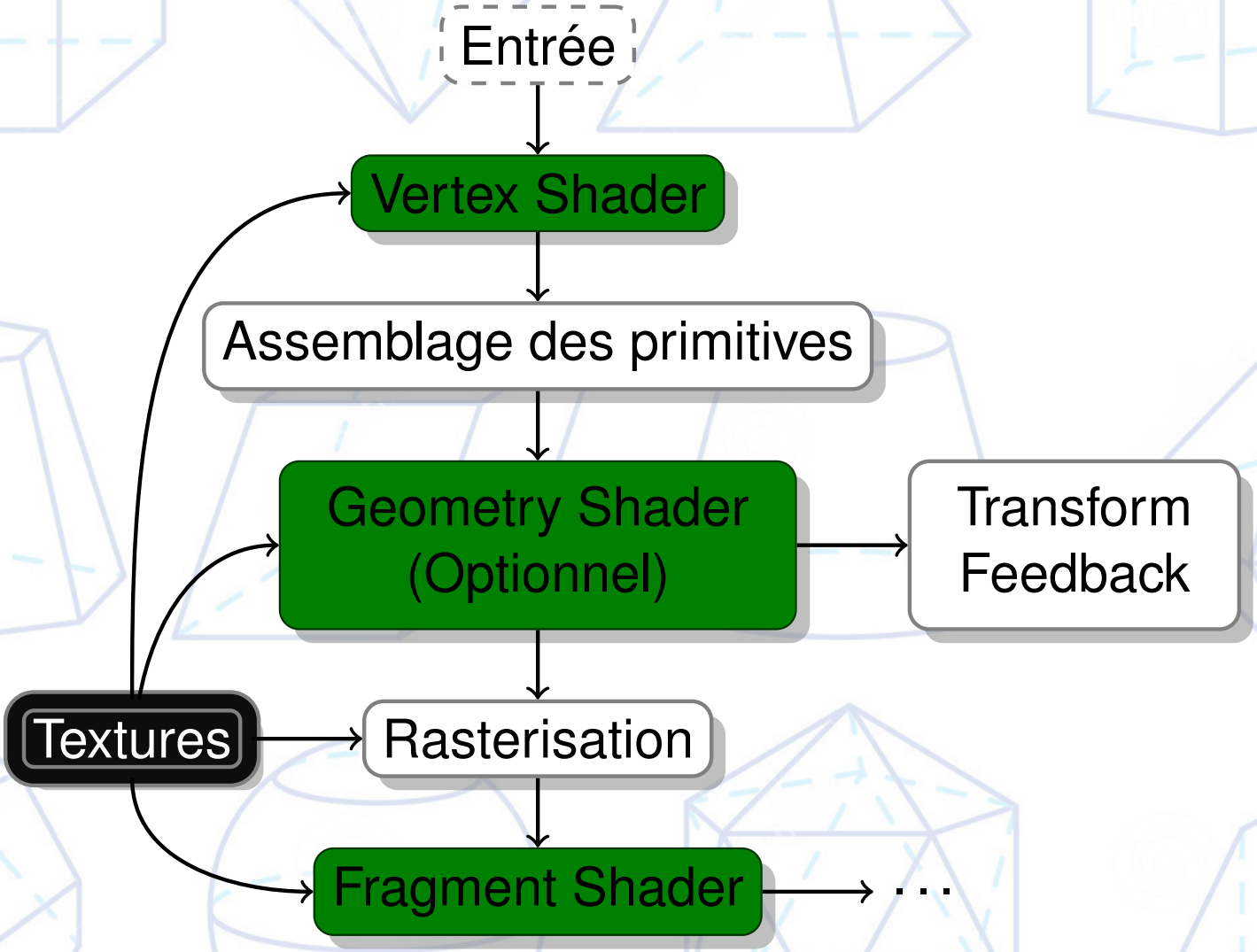
Un nouvel étage

- Après le Vertex Shader
- Avant la rasterization (le découpage en fragments/pixels)
- Avant un éventuel TransformFeedback
- Permet de changer, de supprimer, ou d'ajouter des primitives.
- Peut utiliser des variables uniformes
- Peut recevoir des variables du *vertex shader* (tableaux)

Un nouvel étage

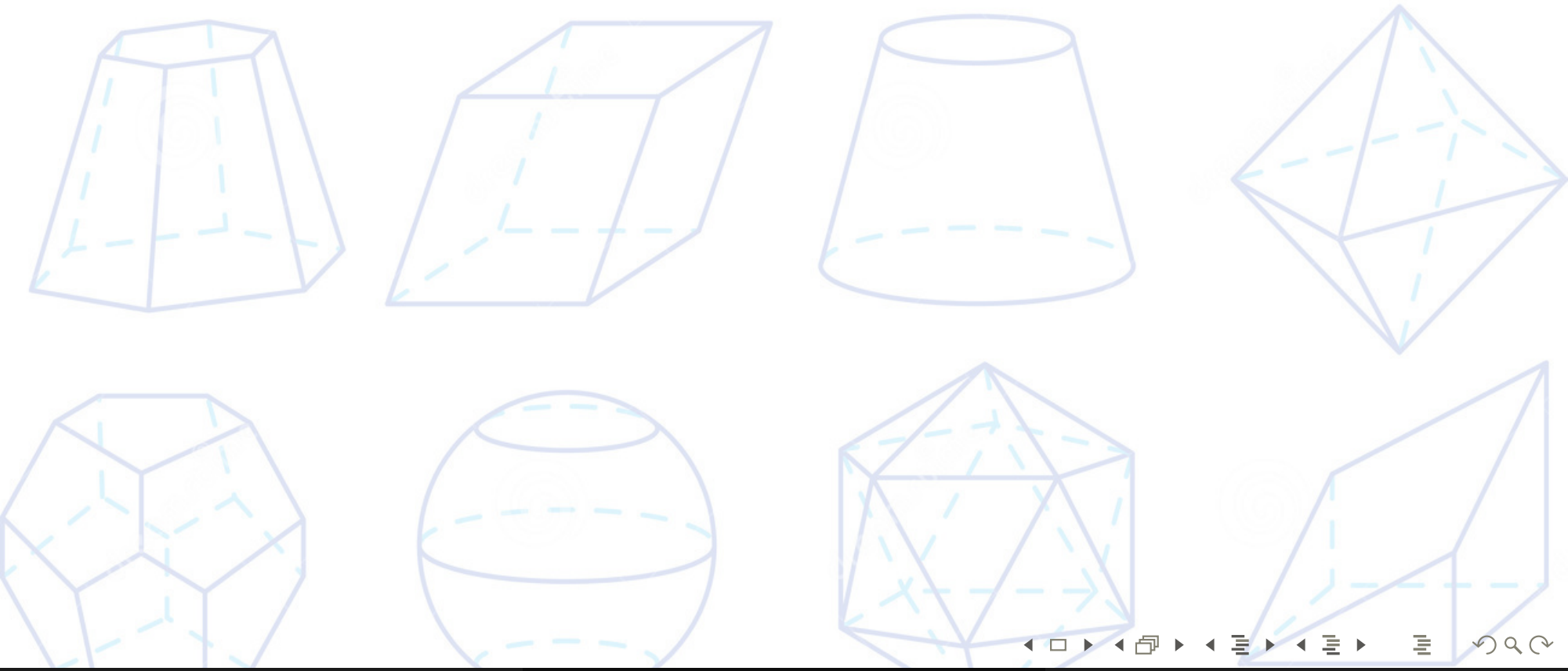
- Après le Vertex Shader
- Avant la rasterization (le découpage en fragments/pixels)
- Avant un éventuel TransformFeedback
- Permet de changer, de supprimer, ou d'ajouter des primitives.
- Peut utiliser des variables uniformes
- Peut recevoir des variables du *vertex shader* (tableaux)
- Peut envoyer des variables (interpolées) au *fragment shader*

Pipeline avec les trois shaders



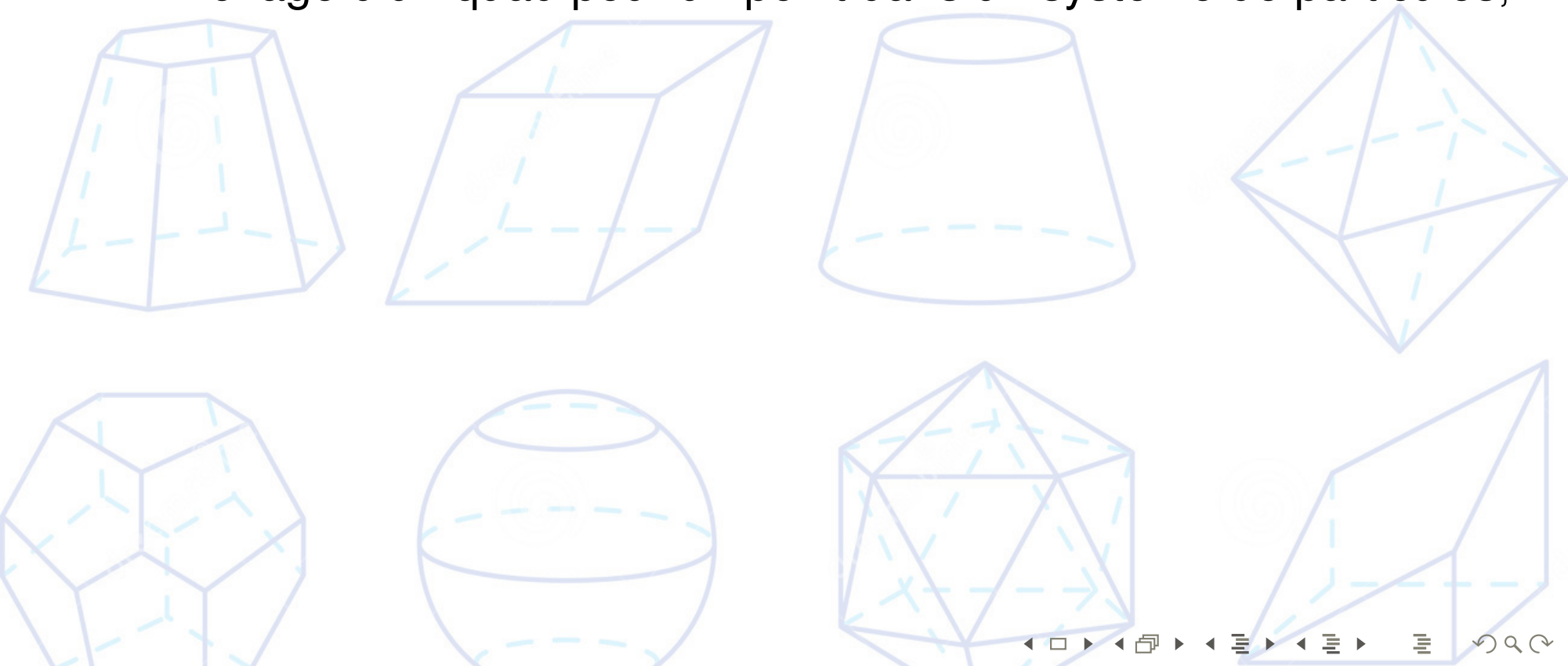
Utilité

- Courbe de Bézier (cheveux, poils),



Utilité

- Courbe de Bézier (cheveux, poils),
- Affichage d'un quad pour un point dans un système de particules,



Utilité

- Courbe de Bézier (cheveux, poils),
- Affichage d'un quad pour un point dans un système de particules,
- Affichage des normales pour le debug,

Utilité

- Courbe de Bézier (cheveux, poils),
- Affichage d'un quad pour un point dans un système de particules,
- Affichage des normales pour le debug,
- Création de cône d'ombre,

Utilité

- Courbe de Bézier (cheveux, poils),
- Affichage d'un quad pour un point dans un système de particules,
- Affichage des normales pour le debug,
- Création de cône d'ombre,
- Cell shading,

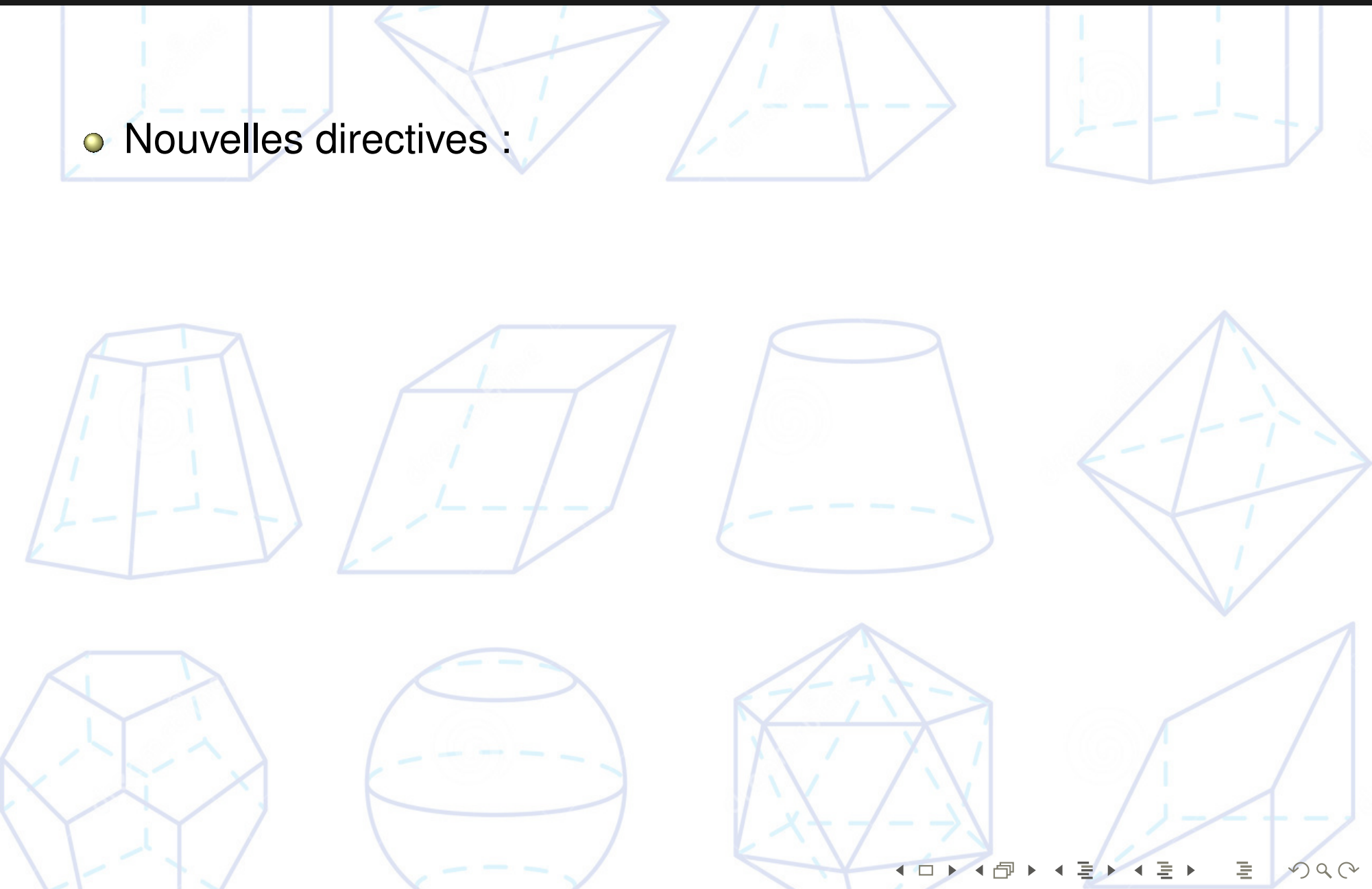
Utilité

- Courbe de Bézier (cheveux, poils),
- Affichage d'un quad pour un point dans un système de particules,
- Affichage des normales pour le debug,
- Création de cône d'ombre,
- Cell shading,
- Marching Cubes,

Utilité

- Courbe de Bézier (cheveux, poils),
- Affichage d'un quad pour un point dans un système de particules,
- Affichage des normales pour le debug,
- Création de cône d'ombre,
- Cell shading,
- Marching Cubes,
- Etc.

- Nouvelles directives :



- Nouvelles directives :
  - **layout**(**primitive**) in ;
    primitive peut être : **points**, **lines**, **triangles**, **lines_adjacency** ou **triangles_adjacency**

- Nouvelles directives :
  - **layout**(**primitive**) in ;
    primitive peut être : **points**, **lines**, **triangles**, **lines_adjacency** ou **triangles_adjacency**
  - **layout**(**primitive**, **max_vertices**=**N**) out;
    primitive peut être : **points**, **line_strip** ou **triangle_strip**

- Nouvelles directives :
  - **layout**(**primitive**) in ;
    primitive peut être : **points**, **lines**, **triangles**, **lines_adjacency** ou **triangles_adjacency**
  - **layout**(**primitive**, **max_vertices**=**N**) out;
    primitive peut être : **points**, **line_strip** ou **triangle_strip**
- Les variables d'entrée (en provenance du *vertex shader*) sont dans des tableaux.

GLSL

- Nouvelles directives :
  - **layout**(**primitive**) in ;
    primitive peut être : **points**, **lines**, **triangles**, **lines_adjacency** ou **triangles_adjacency**
  - **layout**(**primitive**, **max_vertices**=**N**) out;
    primitive peut être : **points**, **line_strip** ou **triangle_strip**
- Les variables d'entrée (en provenance du *vertex shader*) sont dans des tableaux.
  - Prédéfini :**gl_in**[**i**].**gl_Position**

- Nouvelles directives :
  - **layout**(**primitive**) in ;
    primitive peut être : **points**, **lines**, **triangles**, **lines_adjacency** ou **triangles_adjacency**
  - **layout**(**primitive**, **max_vertices**=**N**) out;
    primitive peut être : **points**, **line_strip** ou **triangle_strip**
- Les variables d'entrée (en provenance du *vertex shader*) sont dans des tableaux.
  - Prédéfini :**gl_in**[**i**].**gl_Position**
  - utilisateur : in **vec3** **vg_normal**[];

- Nouvelles directives :
  - **layout**(**primitive**) in ;
    primitive peut être : **points**, **lines**, **triangles**, **lines_adjacency** ou **triangles_adjacency**
  - **layout**(**primitive**, **max_vertices**=**N**) out;
    primitive peut être : **points**, **line_strip** ou **triangle_strip**
- Les variables d'entrée (en provenance du *vertex shader*) sont dans des tableaux.
  - Prédéfini :**gl_in**[**i**].**gl_Position**
  - utilisateur : in **vec3** **vg_normal**[];
- Nouvelles fonctions :

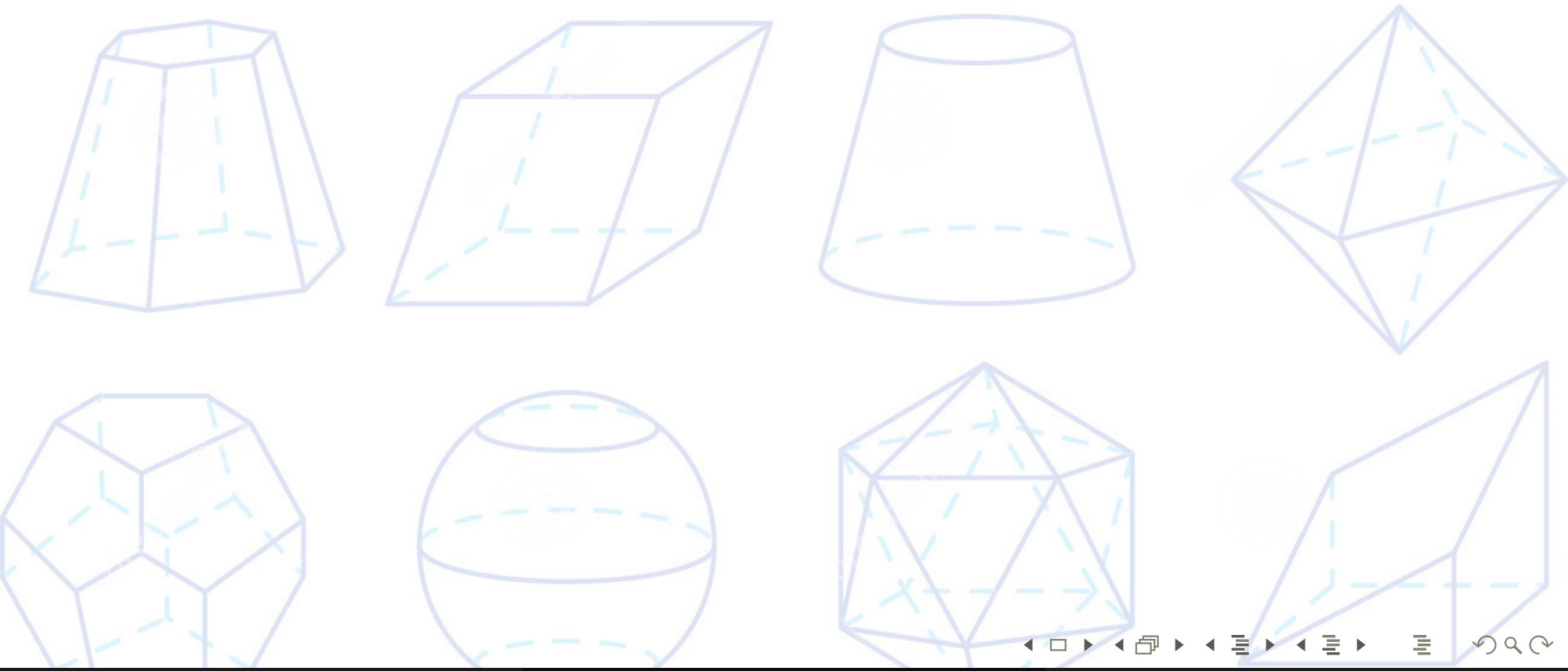
GLSL

- Nouvelles directives :
  - **layout**(**primitive**) in ;
    primitive peut être : **points**, **lines**, **triangles**, **lines_adjacency** ou **triangles_adjacency**
  - **layout**(**primitive**, **max_vertices**=**N**) out;
    primitive peut être : **points**, **line_strip** ou **triangle_strip**
- Les variables d'entrée (en provenance du *vertex shader*) sont dans des tableaux.
  - Prédéfini :**gl_in**[**i**].**gl_Position**
  - utilisateur : in **vec3** **vg_normal**[];
- Nouvelles fonctions :
  - **EmitVertex**(); pour chaque Vertex (position, normale, etc.)

- Nouvelles directives :
  - **layout**(**primitive**) in ;
    primitive peut être : **points**, **lines**, **triangles**, **lines_adjacency** ou **triangles_adjacency**
  - **layout**(**primitive**, **max_vertices**=**N**) out;
    primitive peut être : **points**, **line_strip** ou **triangle_strip**
- Les variables d'entrée (en provenance du *vertex shader*) sont dans des tableaux.
  - Prédéfini :**gl_in**[**i**].**gl_Position**
  - utilisateur : in **vec3** **vg_normal**[];
- Nouvelles fonctions :
  - **EmitVertex**(); pour chaque Vertex (position, normale, etc.)
  - **EndPrimitive**(); pour terminer une primitive.

Exemple : affichage de normales

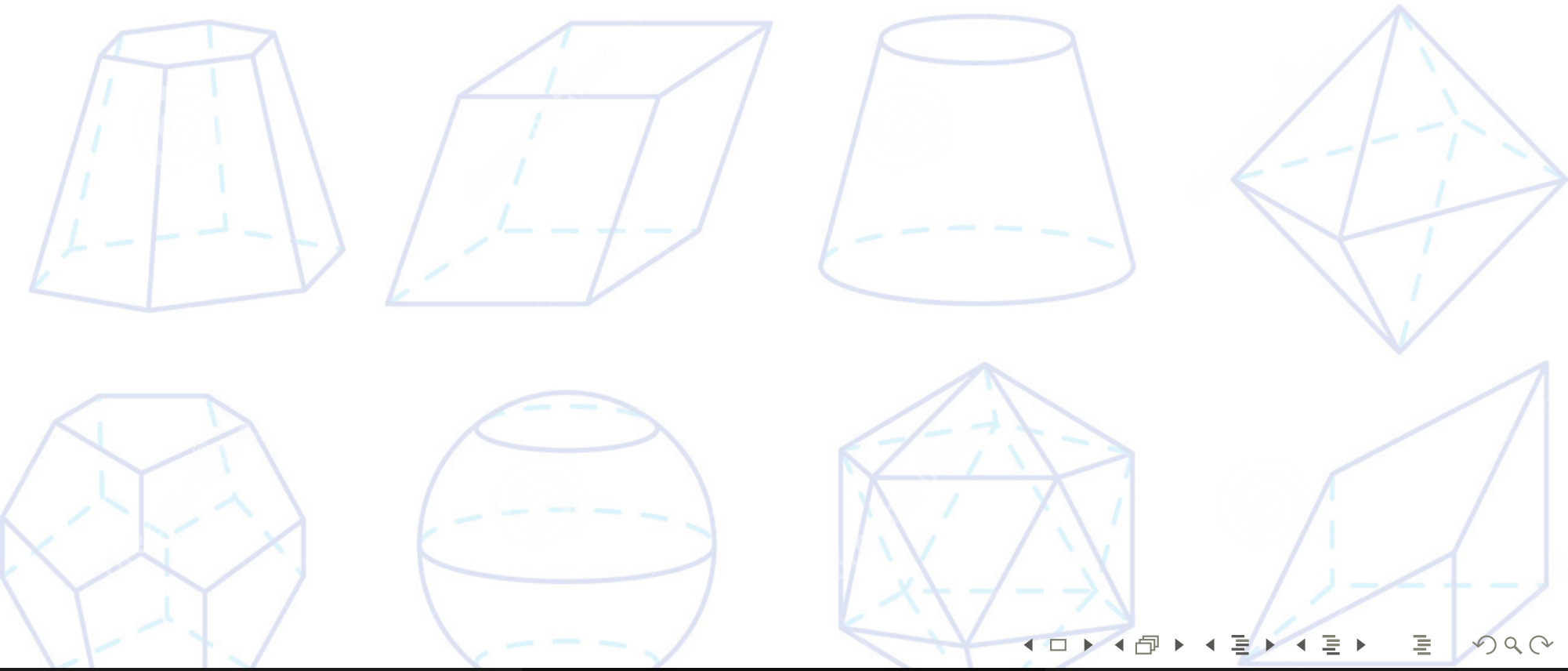
```
1 #version 330
```



EXEMPLE : AFFICHAGE DE NORMALES

```
1 #version 330
```

```
1 layout(triangles) in;
2 layout(line_strip, max_vertices = 6) out;
```



Exemple : affichage de normales

```
1 #version 330
```

```
1 layout(triangles) in;
2 layout(line_strip, max_vertices = 6) out;
```

```
1 in vec3 normal[]; // out vec3 normal; dans le vertex shader
```

EXEMPLE : AFFICHAGE DE NORMALES

```
#version 330
```

```
layout(triangles) in;
layout(line_strip, max_vertices = 6) out;
```

```
in vec3 normal[]; // out vec3 normal; dans le vertex shader
```

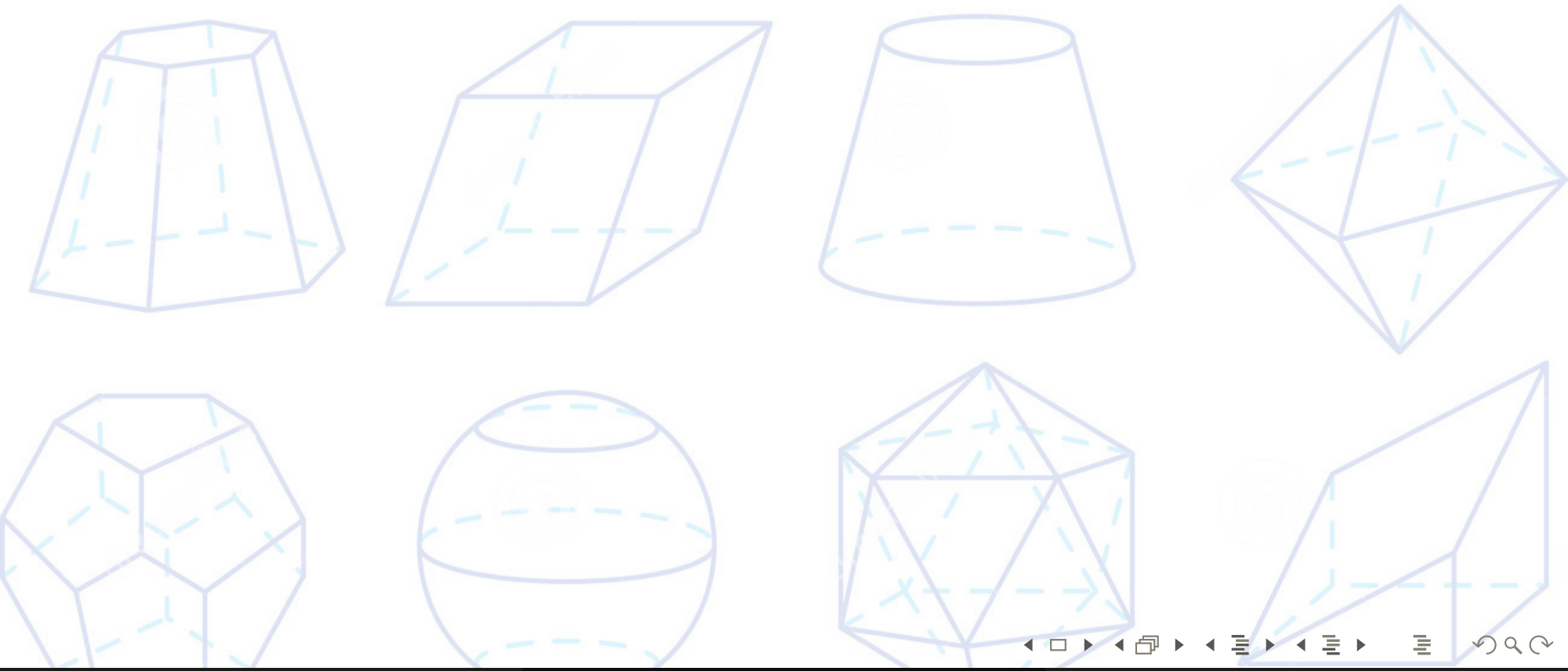
```
void main(void)
{
  for (int i = 0; i < 3; i++)
    {
      gl_Position = gl_in[i].gl_Position;
      EmitVertex();
      gl_Position = gl_in[i].gl_Position + vec4(normal[i], 1.0);
      EmitVertex();

      EndPrimitive();
    }
}
```

Transparence et geometry shader – Plan

Blending

- Pas de vraie transparence



Blending

- Pas de vraie transparence
- Mélange entre couleur/alpha du pixel dans le framebuffer (dest), et couleur/alpha du pixel courant (source)

Blending

- Pas de vraie transparence
- Mélange entre couleur/alpha du pixel dans le framebuffer (dest), et couleur/alpha du pixel courant (source)
- glEnable(**GL_BLEND**)

Blending

- Pas de vraie transparence
- Mélange entre couleur/alpha du pixel dans le framebuffer (dest), et couleur/alpha du pixel courant (source)
- glEnable(**GL_BLEND**)
- glBlendFunc(**facteur_source**, **facteur_destination**)

Blending

- Pas de vraie transparence
- Mélange entre couleur/alpha du pixel dans le framebuffer (dest), et couleur/alpha du pixel courant (source)
- glEnable(**GL_BLEND**)
- glBlendFunc(**facteur_source**, **facteur_destination**)
- $C_f = f_s \cdot C_s + f_d \cdot C_d$

Formules

f_s et f_d peuvent prendre les valeurs suivantes :

Valeur	Formule associée
GL_ZERO	$(0, 0, 0)$
GL_ONE	$(1, 1, 1)$
GL_SRC_COLOR	(R_s, V_s, B_s)
GL_ONE_MINUS_SRC_COLOR	$(1, 1, 1) - (R_s, V_s, B_s)$
GL_DST_COLOR	(R_d, V_d, B_d)
GL_ONE_MINUS_DST_COLOR	$(1, 1, 1) - (R_d, V_d, B_d)$
GL_SRC_ALPHA	(A_s, A_s, A_s)
GL_ONE_MINUS_SRC_ALPHA	$(1, 1, 1) - (A_s, A_s, A_s)$
GL_DST_ALPHA	(A_d, A_d, A_d)
GL_ONE_MINUS_DST_ALPHA	$(1, 1, 1) - (A_d, A_d, A_d)$
GL_SRC_ALPHA_SATURATE	$(i, i, i), i = \min(A_s, 1 - A_d)$
GL_CONSTANT_COLOR	(R_c, V_c, B_c)
GL_ONE_MINUS_CONSTANT_COLOR	$(1, 1, 1) - (R_c, V_c, B_c)$
GL_CONSTANT_ALPHA	(A_c, A_c, A_c)
GL_ONE_MINUS_CONSTANT_ALPHA	$(1, 1, 1) - (A_c, A_c, A_c, A_c)$

Couleur et équation

- Paramétrage de la couleur :

  void glBlendColor(**GLclampf red**, **GLclampf green**, **GLclampf blue**, **GLclampf alpha**);

Couleur et équation

- Paramétrage de la couleur :

  void glBlendColor(**GLclampf red**, **GLclampf green**, **GLclampf blue**, **GLclampf alpha**);
- Changement de l'équation : **void** glBlendEquation(**GLenum mode**);
  - **GL_FUNC_ADD**,
  - **GL_FUNC_SUBTRACT**,
  - **GL_FUNC_REVERSE_SUBTRACT**,
  - **GL_MIN**,
  - **GL_MAX**.

Séparation de l'alpha

- Avoir des facteurs ou une équation différente pour la composante *alpha*

Séparation de l'alpha

- Avoir des facteurs ou une équation différente pour la composante *alpha*
- **void** glBlendEquationSeparate(**GLenum modeRGB**, **GLenum modeAlpha**);

Séparation de l'alpha

- Avoir des facteurs ou une équation différente pour la composante *alpha*
- **void** glBlendEquationSeparate(**GLenum modeRGB**, **GLenum modeAlpha**);

```
1 void glBlendFuncSeparate(GLenum srcRGB, GLenum dstRGB,
2                          GLenum srcAlpha, GLenum dstAlpha);
```