Dessiné avec AGE

Bonjour, voici un petit tutoriel destiné à vous permettre d'utilisé la surcouche OpenGL du projet AGE, ceci afin de pouvoir rendre à l'écran ce que vous souhaité.

Accédé au rendering :

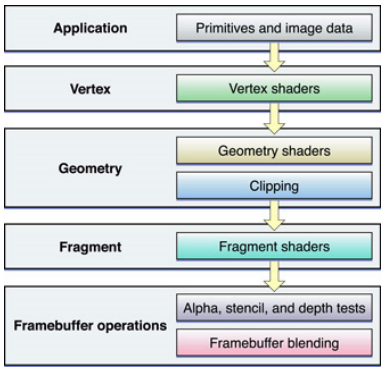
Le rendu est accessible via la dépendance de type RenderManager. Cette classe permet l'instanciation d'un objet unique englobant toute les problématiques de shading, management des ressources graphique et dessin.

Important : Toute opération à destination du rendu se doit d'être effectuée via cette classe.

Les Shaders :

Les shaders sont des petits bouts de code exécuter par la carte graphique afin dessiner des éléments 2D ou 3D sur votre écran.

La surcouche de AGE vous donne accès à 3 unité de traitement d'un shader : le Vertex Shader, le fragment Shader, et géométrie Shader. Il vous permet aussi d'utilisé le compute shader récemment intégré à OpenGL.



Créé un shader avec AGE :

```
Key<Shader> addComputeShader(std::string const &compute);
Key<Shader> addShader(std::string const &vert, std::string const &frag);
Key<Shader> addShader(std::string const &geometry, std::string const &vert, std::string const &frag);
```

Voici-ci trois classe ayant pour objectif de créé un shader dans le contexte AGE. L'objet renvoyé de type Key<Shader> est une clef permettant de manipulé l'instance nouvellement crée.

Les ressources de shader :

AGE offre la possibilité d'envoyer des données au shader. Nous distinguons 3 types de données différentes.

- Les Uniform
- Les Samplers
- Les Interfaces Block

Uniform :

Ils sont manipulables au travers des fonctions suivantes

```
Key<Uniform> addShaderUniform(Key<Shader> const &shader, std::string const &flag);
Key<Uniform> addShaderUniform(Key<Shader> const &shader, std::string const &flag, glm::mat4 const &value);
Key<Uniform> addShaderUniform(Key<Shader> const &shader, std::string const &flag, glm::mat3 const &value);
Key<Uniform> addShaderUniform(Key<Shader> const &shader, std::string const &flag, float value);
Key<Uniform> addShaderUniform(Key<Shader> const &shader, std::string const &flag, glm::vec4 const &value);
Key<Uniform> getShaderUniform(Key<Shader> const &shader, size_t index);
RenderManager &setShaderUniform(Key<Shader> const &shader, Key<Uniform> const &key, glm::mat4 const &mat4);
RenderManager &setShaderUniform(Key<Shader> const &shader, Key<Uniform> const &key, glm::vec4 const &vec4);
RenderManager &setShaderUniform(Key<Shader> const &shader, Key<Uniform> const &key, float v);
RenderManager &setShaderUniform(Key<Shader> const &shader, Key<Uniform> const &key, glm::mat3 const &mat3);
```

Leur création est similaire au système utilisé pour les shaders. Leur setting passe au travers d'une simple surcharge de la fonction addShaderUniform ou setShaderUniform.

Pour le setting comme pour la création une clef de shader est demandée afin de spécifier à qu'elle kernel appartiens l'Uniform.

Sampler :

```
Key<Sampler> addShaderSampler(Key<Shader> const &shader, std::string const &flag);
Key<Sampler> getShaderSampler(Key<Shader> const &shader, size_t index);
RenderManager &setShaderSampler(Key<Shader> const &shader, Key<Sampler> const &key, Key<Texture> const &keytexture);
```

Les samplers fonctionnent de la même manière que les Uniform, seulement ils nécessitent d'être setter par l'intermédiaire d'une clef de texture. (Voir chapitre Texture).

Interface Block:

```
Key<InterfaceBlock> addShaderInterfaceBlock(Key<Shader> const &shader, std::string const &flag, Key<UniformBlock> const &keyUniformBlock);
Key<InterfaceBlock> getShaderInterfaceBlock(Key<Shader> const &shader, size_t index);
```

Les interfaces block servent à bind les Uniform block au shader, à terme elles Pourront accepter les Shaders Storage Object et plus encore.