Дипломна робота

Створено системою Doxygen 1.9.1

1 GNU LESSER GENERAL PUBLIC LICENSE

Version 3, 29 June 2007

This version of the GNU Lesser General Public License incorporates the terms and conditions of version 3 of the GNU General Public License, supplemented by the additional permissions listed below.

1.0.0.1 0. Additional Definitions. As used herein, "this License" refers to version 3 of the GNU Lesser General Public License, and the "GNU GPL" refers to version 3 of the GNU General Public License.

"The Library" refers to a covered work governed by this License, other than an Application or a Combined Work as defined below.

An "Application" is any work that makes use of an interface provided by the Library, but which is not otherwise based on the Library. Defining a subclass of a class defined by the Library is deemed a mode of using an interface provided by the Library.

A "Combined Work" is a work produced by combining or linking an Application with the Library. The particular version of the Library with which the Combined Work was made is also called the "Linked Version".

The "Minimal Corresponding Source" for a Combined Work means the Corresponding Source for the Combined Work, excluding any source code for portions of the Combined Work that, considered in isolation, are based on the Application, and not on the Linked Version.

The "Corresponding Application Code" for a Combined Work means the object code and/or source code for the Application, including any data and utility programs needed for reproducing the Combined Work from the Application, but excluding the System Libraries of the Combined Work.

1.0.0.2 1. Exception to Section 3 of the GNU GPL. You may convey a covered work under sections 3 and 4 of this License without being bound by section 3 of the GNU GPL.

1.0.0.3 2. Conveying Modified Versions. If you modify a copy of the Library, and, in your modifications, a facility refers to a function or data to be supplied by an Application that uses the facility (other than as an argument passed when the facility is invoked), then you may convey a copy of the modified version:

- a) under this License, provided that you make a good faith effort to ensure that, in the event an Application does not supply the function or data, the facility still operates, and performs whatever part of its purpose remains meaningful, or
- b) under the GNU GPL, with none of the additional permissions of this License applicable to that copy.

1.0.0.4 3. Object Code Incorporating Material from Library Header Files. The object code form of an Application may incorporate material from a header file that is part of the Library. You may convey such object code under terms of your choice, provided that, if the incorporated material is not limited to numerical parameters, data structure layouts and accessors, or small macros, inline functions and templates (ten or fewer lines in length), you do both of the following:

- a) Give prominent notice with each copy of the object code that the Library is used in it and that the Library and its use are covered by this License.
- b) Accompany the object code with a copy of the GNU GPL and this license document.

1.0.0.5 4. Combined Works. You may convey a Combined Work under terms of your choice that, taken together, effectively do not restrict modification of the portions of the Library contained in the Combined Work and reverse engineering for debugging such modifications, if you also do each of the following:

- a) Give prominent notice with each copy of the Combined Work that the Library is used in it and that the Library and its use are covered by this License.
- b) Accompany the Combined Work with a copy of the GNU GPL and this license document.
- c) For a Combined Work that displays copyright notices during execution, include the copyright notice for the Library among these notices, as well as a reference directing the user to the copies of the GNU GPL and this license document.
- d) Do one of the following:
  - 0) Convey the Minimal Corresponding Source under the terms of this License, and the Corresponding Application Code in a form suitable for, and under terms that permit, the user to recombine or relink the Application with a modified version of the Linked Version to produce a modified Combined Work, in the manner specified by section 6 of the GNU GPL for conveying Corresponding Source.
  - 1) Use a suitable shared library mechanism for linking with the Library. A suitable mechanism is one that (a) uses at run time a copy of the Library already present on the user's computer system, and (b) will operate properly with a modified version of the Library that is interface-compatible with the Linked Version.
- e) Provide Installation Information, but only if you would otherwise be required to provide such information under section 6 of the GNU GPL, and only to the extent that such information is necessary to install and execute a modified version of the Combined Work produced by recombining or relinking the Application with a modified version of the Linked Version. (If you use option 4d0, the Installation Information must accompany the Minimal Corresponding Source and Corresponding Application Code. If you use option 4d1, you must provide the Installation Information in the manner specified by section 6 of the GNU GPL for conveying Corresponding Source.)

1.0.0.6 5. Combined Libraries. You may place library facilities that are a work based on the Library side by side in a single library together with other library facilities that are not Applications and are not covered by this License, and convey such a combined library under terms of your choice, if you do both of the following:

- a) Accompany the combined library with a copy of the same work based on the Library, uncombined with any other library facilities, conveyed under the terms of this License.
- b) Give prominent notice with the combined library that part of it is a work based on the Library, and explaining where to find the accompanying uncombined form of the same work.

1.0.0.7 6. Revised Versions of the GNU Lesser General Public License. The Free Software Foundation may publish revised and/or new versions of the GNU Lesser General Public License from time to time. Such new versions will be similar in spirit to the present version, but may differ in detail to address new problems or concerns.

Each version is given a distinguishing version number. If the Library as you received it specifies that a certain numbered version of the GNU Lesser General Public License "or any later version" applies to it, you have the option of following the terms and conditions either of that published version or of any later version published by the Free Software Foundation. If the Library as you received it does not specify a version number of the GNU Lesser General Public License, you may choose any version of the GNU Lesser General Public License ever published by the Free Software Foundation.

If the Library as you received it specifies that a proxy can decide whether future versions of the GNU Lesser General Public License shall apply, that proxy's public statement of acceptance of any version is permanent authorization for you to choose that version for the Library.

2 Алфавітний покажчик простору імен

2.1 Простір імен

Повний список просторів імен.

3 Ієрархічний покажчик класів

3.1 Ієрархія класів

Список успадкувань впорядковано наближено до алфавіту

4 Алфавітний покажчик класів

4.1 Класи

Класи, структури, об'єднання та інтерфейси з коротким описом.

5 Покажчик файлв

5.1 Файли

Повний список файлів.

6 Опис простору імен

6.1 Простір імен OSDO

Неймспейс бібліотеки osdo

Класи

- class vector
  Вектор що не змінює свій розмір.

6.1.1 Детальний опис

Неймспейс бібліотеки osdo

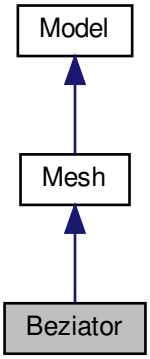
7 Класи

7.1 Клас Beziator

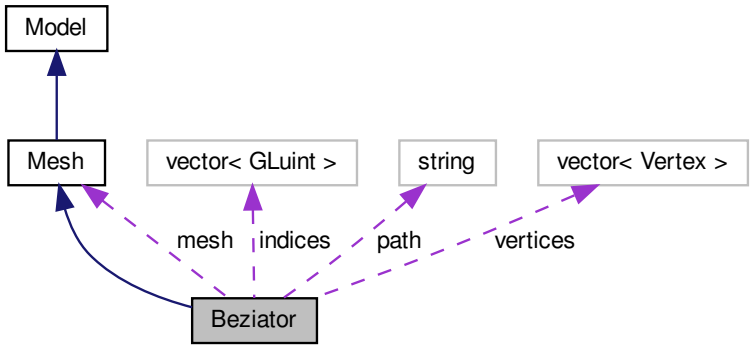
Клас який зберігає та оброблює модель утворену через поверхні Безьє.

#include <beziator.h>

Схема успадкувань для Beziator



Діаграма зв'язків класу Beziator:



Загальнодоступні типи

- typedef surfacei_t * surfaces_vector

  Тип позначаючий вказівнкик на массив з поверхнями Безьє.

Загальнодоступні елементи

- Beziator (const string &path)

  Конструктор до Beziator, який зберігає шлях до файлу з моделлю.
- ~Beziator () override
- bool init ()

  Завантажує модель у пам'ять.

- void draw (Shader &shader, bool pre_generated) override

  Відображує модель.
- void generate (size_t d=8) override

  Генерує деталізований меш моделі.
- bool save ()

  Зберігає модель у файл, вказаний у полі path.
- void rotate (size_t i)

  Інвертує порядок індесів поверхні, щоб нормалі дивилися у протилежний бік.
- vector< Vertex > * get_vertices () override

  Видає список вершин моделі.

Захищені дані

- const string path

  Шлях до файлу у якому зберігається модель.
- Mesh mesh

  Згенерований за допомогою CPU меш моделі.
- vector< Vertex > vertices

  Масив вершин/вузлів моделі.
- vector< GLuint > indices

  Масив індексів, що утворюють поверхні Безье.

7.1.1 Детальний опис

Клас який зберігає та оброблює модель утвореню через поверхні Безьє.

Див. визначення в файлі beziator.h, рядок 22

7.1.2 Опис типів користувача

7.1.2.1 surfaces_vector typedef surfacei_t* Beziator::surfaces_vector

Тип позначаючий вказівнкик на массив з поверхнями Безьє.

Див. визначення в файлі beziator.h, рядок 27

7.1.3 Конструктор(и)

7.1.3.1 Beziator() Beziator::Beziator (const string & path)

Конструктор до Beziator, який зберігає шлях до файлу з моделлю.

Обов'язково потібно запустити метод Beziator::init для того щоб завантажити модель у пам'ять.

Аргументи

path	Шлях до файлу у якому зберігається модель.

Див. визначення в файлі beziator.cpp, рядок 18

7.1.3.2 ~Beziator() Beziator::~Beziator ( ) [override]

Див. визначення в файлі beziator.cpp, рядок 58

7.1.4 Опис методів компонент

7.1.4.1 draw() void Beziator::draw (Shader & shader, bool pre_generated) [override], [virtual]

Відображує модель.

За допомогою флагу pre_generated можна задати яким чином потібно відображати, якщо задати false, то у буде використаний меш із поверхнями Безье 4x4, а якщо задано true, то відобразиться сгенерований деталізований меш моделі.

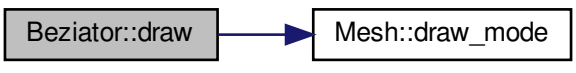
Аргументи

shader	Шейдер який використовуєтсья для відображення моделі.
pre_generated	Флаг, який позначає який з мешів відображати.

Переозначення з Model.

Див. визначення в файлі beziator.cpp, рядок 61

Граф всіх викликів цієї функції:



7.1.4.2 generate() void Beziator::generate (size_t d = 8) [override], [virtual]

Генерує деталізований меш моделі.

Ступінь деталізаії d позначає скільки вершин буде створено по двом осям, за заммовчанням задано 8, таким чином поверхня буде складатися з 8x8=64 вершини.

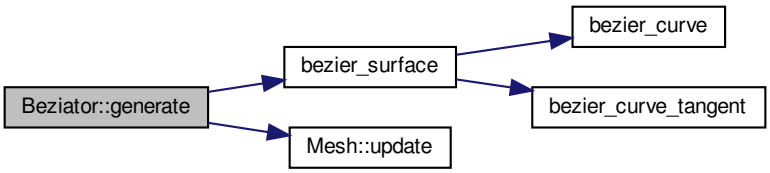
Аргументи

d	ступінь деталізації.

Переозначення з Model.

Див. визначення в файлі beziator.cpp, рядок 136

Граф всіх викликів цієї функції:



7.1.4.3 get_vertices() vector< Vertex > * Beziator::get_vertices ( ) [override], [virtual]

Видає список вершин моделі.

Повертає

Вказівник на поле vertices.

Переозначення з Model.

Див. визначення в файлі beziator.cpp, рядок 287

7.1.4.4 init() bool Beziator::init ( )

Завантажує модель у пам'ять.

Повертає

Статус, чи успішно була завантажена модель.

Див. визначення в файлі beziator.cpp, рядок 20

Граф всіх викликів цієї функції:

7.1.4.5 rotate() void Beziator::rotate (size_t i)

Інвертує порядок індесів поверхні, щоб нормалі дивилися у протилежний бік.

Аргументи

i	номер поверхні.

Див. визначення в файлі beziator.cpp, рядок 277

7.1.4.6 save() bool Beziator::save ( )

Зберігає модель у файл, вказаний у полі path.

Повертає

Статус зберігання файлу.

Див. визначення в файлі beziator.cpp, рядок 110

7.1.5 Компонентні дані

7.1.5.1 indices vector<GLuint> Beziator::indices [protected]

Масив індексів, що утворюють поверхні Безье.

Індекси розташовані у масиві по 16 елементів, які утворюють поверхню з контрольними точками 4x4. Масив легко інтерпретуєтсья у surfaces_vector:

```
surfacei_t *surfaces = reinterpret_cast<surfacei_t*>(indices.data());
```

Див. визначення в файлі beziator.h, рядок 52

7.1.5.2 mesh Mesh Beziator::mesh [protected]

Згенерований за допомогою CPU меш моделі.

Див. визначення в файлі beziator.h, рядок 36

7.1.5.3 path const string Beziator::path [protected]

Шлях до файлу у якому зберігається модель.

Див. визначення в файлі beziator.h, рядок 32

7.1.5.4 vertices vector<Vertex> Beziator::vertices [protected]

Масив вершин/вузлів моделі.

Див. визначення в файлі beziator.h, рядок 42

Документація цих класів була створена з файлів:

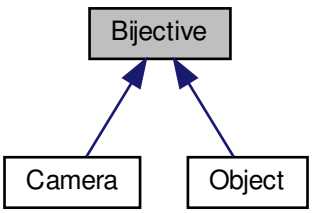
- osdo/beziator.h
- osdo/beziator.cpp

7.2 Клас Bijective

Інтерфейс до об'єктів, що можуть можуть бути переміщені та повернуті у просторі.

#include <bijective.h>

Схема успадкувань для Bijective



Загальнодоступні елементи

- virtual ~Bijective ()
- virtual void get_position (vec4 position)

  Забирає поточну позицію об'єкта у просторі.
- virtual void set_position (vec4 position)

  Задає нову позицію об'єкта у просторі.
- virtual void get_rotation (vec3 rotation)

  Забирає поточний нахил об'єкта.
- virtual void set_rotation (vec3 rotation)

  Задає новий нахил об'єкта.
- virtual void get_animation (vec3 rotation)

  Забирає поточну анімацію обернення об'єкта.
- virtual void set_animation (vec3 rotation)

  Задає нову анімацію обернення об'єкта.
- virtual void get_mat4 (mat4 matrix)

  Забирає матрицю лінійних перетворень над об'єктом.
- virtual void translate (vec3 distances, float delta_time)

  Переміщує об'єкт у просторі.
- virtual void rotate (enum coord_enum coord, float delta_time)

  Обертає об'єкт.
- virtual void rotate_all (vec3 angles)

  Обернути об'єкт по всім осям.
- virtual void add_animation (vec3 angles, float delta_time)

  Додає швидкість анімації обертання об'єкту.

7.2.1 Детальний опис

Інтерфейс до об'єктів, що можуть можуть бути переміщені та повернуті у просторі.

Див. визначення в файлі bijective.h, рядок 13

7.2.2 Конструктор(и)

7.2.2.1 ~Bijective() virtual Bijective::~Bijective ( ) [inline], [virtual]

Див. визначення в файлі bijective.h, рядок 15

7.2.3 Опис методів компонент

7.2.3.1 add_animation() virtual void Bijective::add_animation (vec3 angles, float delta_time) [inline], [virtual]

Додає швидкість анімації обертання об'єкту.

Аргументи

in	angles	вектор швидкостей анімацій обертання по трьом осям
in	delta_time	скільки часу пройшло з останнього кадру

Переозначається в Object і Camera.

Див. визначення в файлі bijective.h, рядок 81

7.2.3.2 get_animation() virtual void Bijective::get_animation (vec3 rotation) [inline], [virtual]

Забирає поточну анімацію обернення об'єкта.

Аргументи

out	rotation	поточна анімація обернення об'єкта

Переозначається в Object і Camera.

Див. визначення в файлі bijective.h, рядок 43

7.2.3.3 get_mat4() virtual void Bijective::get_mat4 (mat4 matrix) [inline], [virtual]

Забирає матрицю лінійних перетворень над об'єктом.

Аргументи

out	matrix	матриця лінійних перетворень

Переозначається в Object і Camera.

Див. визначення в файлі bijective.h, рядок 54

7.2.3.4 get_position() virtual void Bijective::get_position (vec4 position) [inline], [virtual]

Забирає поточну позицію об'єкта у просторі.

Аргументи

out	position	поточна позицію об'єкта

Переозначається в Object і Camera.

Див. визначення в файлі bijective.h, рядок 21

7.2.3.5 get_rotation() virtual void Bijective::get_rotation (vec3 rotation) [inline], [virtual]

Забирає поточний нахил об'єкта.

Аргументи

out	rotation	поточний нахил об'єкта

Переозначається в Object і Camera.

Див. визначення в файлі bijective.h, рядок 32

7.2.3.6 rotate() virtual void Bijective::rotate (enum coord_enum coord, float delta_time) [inline], [virtual]

Обертає об'єкт.

Аргументи

in	coord	позначає координатну вісь навколо якої обертати
in	delta_time	скільки часу пройшло з останнього кадру

Переозначається в Object і Camera.

Див. визначення в файлі bijective.h, рядок 70

7.2.3.7 rotate_all() virtual void Bijective::rotate_all (vec3 angles) [inline], [virtual]

Обернути об'єкт по всім осям.

Аргументи

in	angles	вектор кутів у радіанах на кожну вісь

Переозначається в Object і Camera.

Див. визначення в файлі bijective.h, рядок 75

7.2.3.8 set_animation() virtual void Bijective::set_animation (vec3 rotation) [inline], [virtual]

Задає нову анімацію обернення об'єкта.

Аргументи

in	rotation	нова анімація обернення об'єкта.

Переозначається в Object і Camera.

Див. визначення в файлі bijective.h, рядок 48

7.2.3.9 set_position() virtual void Bijective::set_position (vec4 position) [inline], [virtual]

Задає нову позицію об'єкта у просторі.

Аргументи

in	position	нова позиція об'єкта у просторі

Переозначається в Object і Camera.

Див. визначення в файлі bijective.h, рядок 26

7.2.3.10 set_rotation() virtual void Bijective::set_rotation (vec3 rotation) [inline], [virtual]

Задає новий нахил об'єкта.

Аргументи

in	rotation	новий нахил об'єкта

Переозначається в Object і Camera.

Див. визначення в файлі bijective.h, рядок 37

7.2.3.11 translate() virtual void Bijective::translate (vec3 distances, float delta_time) [inline], [virtual]

Переміщує об'єкт у просторі.

Переміщує об'єкт у просторі на відстані з аргументу distances, де кожне значення вектору позначає відстань відповідної осі.

Аргументи

in	distances	відстані переміщення по осям
in	delta_time	скільки часу пройшло з останнього кадру

Переозначається в Object і Camera.

Див. визначення в файлі bijective.h, рядок 64

Документація цього класу була створена з файлу:

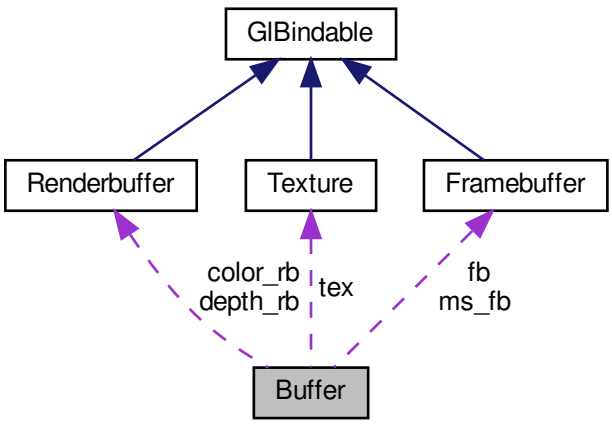
- osdo/bijective.h

7.3 Клас Buffer

Буфер, у якому відбувається рендеринг у текстуру.

#include <buffer.h>

Діаграма зв'язків класу Buffer:



Загальнодоступні елементи

- bool pre_render (GLsizei size[2])
  Підготовка до рендеренгу.
- void post_render (GLsizei size[2])
  Генерація текстури з утвореного кадру.
- const Texture & get_tex ()
  Забирає текстуру у яку проводився рендеринг.

Приватні дані

- Texture tex

  Текстура у яку відбувається рендеринг.
- Renderbuffer color_rb

  Буфер кольорів для рендеренгу.
- Renderbuffer depth_rb

  Глибинний буфер для рендеренгу.
- Framebuffer ms_fb

  Буфер утвореного кадру рендеренгу із згладжуванням.
- Framebuffer fb

  Кінцевий буфер утвореного кадру.

7.3.1 Детальний опис

Буфер, у якому відбувається рендеринг у текстуру.

Див. визначення в файлі buffer.h, рядок 16

7.3.2 Опис методів компонент

7.3.2.1 get_tex() const Texture & Buffer::get_tex ( )

Забирає текстуру у яку проводився рендеринг.

Повертає

текстура у яку проводився рендеринг

Див. визначення в файлі buffer.cpp, рядок 35

7.3.2.2 post_render() void Buffer::post_render (GLsizei size[2])

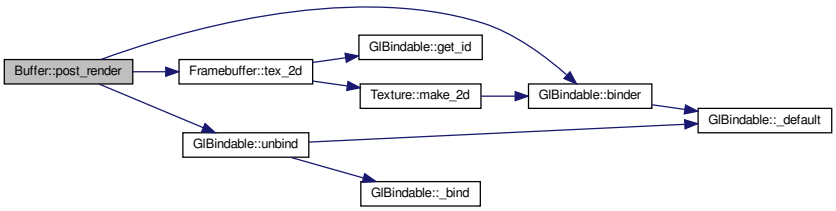
Генерація текстури з утвореного кадру.

Аргументи

in	size	ширина та висота кадру.

Див. визначення в файлі buffer.cpp, рядок 21

Граф всіх викликів цієї функції:



7.3.2.3 pre_render() bool Buffer::pre_render (GLsizei size[2])

Підгодовка до рендеренгу.

Аргументи

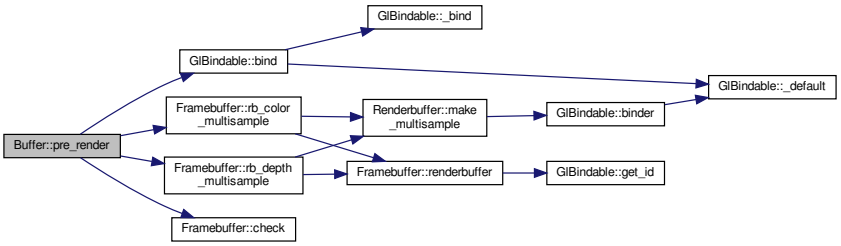
in	size	ширина та висота кадру.

Повертає

статус успішності підготовки буферів до рендеренгу.

Див. визначення в файлі buffer.cpp, рядок 6

Граф всіх викликів цієї функції:



7.3.3 Компонентні дані

7.3.3.1 color_rb Renderbuffer Buffer::color_rb [private]

Буфер кольорів для рендеренгу.

Див. визначення в файлі buffer.h, рядок 24

7.3.3.2 depth_rb Renderbuffer Buffer::depth_rb [private]

Глибинний буфер для рендеренгу.

Див. визначення в файлі buffer.h, рядок 28

7.3.3.3 fb Framebuffer Buffer::fb [private]

Кінцевий буфер утвореного кадру.

Див. визначення в файлі buffer.h, рядок 36

7.3.3.4 ms_fb Framebuffer Buffer::ms_fb [private]

Буфер утвореного кадру рендеренгу із згладжуванням.

Див. визначення в файлі buffer.h, рядок 32

7.3.3.5 tex Texture Buffer::tex [private]

Текстура у яку відбувається рендеринг.

Див. визначення в файлі buffer.h, рядок 20

Документація цих класів була створена з файлів:

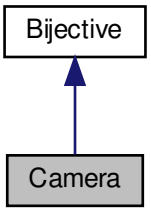
- osdo/buffer.h
- osdo/buffer.cpp

7.4 Клас Camera

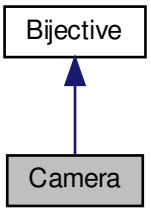
Клас камери, якою можна маніпулювати у сцені.

#include <camera.h>

Схема успадкувань для Camera



Діаграма зв'язків класу Camera:



Загальнодоступні елементи

- Camera ()
- void get_position (vec4 position) override
  Забирає поточну позицію камери у просторі.
- void set_position (vec4 position) override
  Задає нову позицію камери у просторі.
- void get_rotation (vec3 rotation) override
  Забирає поточний нахил камери.
- void set_rotation (vec3 rotation) override
  Задає новий нахил камери.
- void get_animation (vec3 animation) override
  Метод реалізований для інтерфейсу Bijective
- void set_animation (vec3 animation) override

Метод реалізований для інтерфейсу Bijective

- void get_mat4 (mat4 matrix) override

  Забирає матрицю лінійних перетворень над камерою.
- void translate (vec3 distances, float delta_time) override

  Переміщує камеру у просторі.
- void rotate (enum coord_enum coord, float delta_time) override

  Обертає камеру.
- void rotate_all (vec3 angles) override

  Обернути камеру по всім осям.
- void add_animation (vec3 angles, float delta_time) override

  Метод реалізований для інтерфейсу Bijective
- void get_direction (vec4 dest)

  Обчислює напрямок перегляду камери.
- void get_rotation_mat4 (mat4 dest)

  Забирає матрицю обертання камери.
- void get_rotation_inv_mat4 (mat4 dest)

  Забирає інвертовану матрицю обертання камери.
- void translate_camera (vec3 distances)

  Переміщує камеру незалежно від часу.
- void rotate_camera (float angle, enum coord_enum coord)

  Обертає камеру по осі незалежно від часу.
- void rotate_all_camera (vec3 angles)

  Обертає камеру по всім осям незалежно від часу.
- void rotate_all_inverse (vec3 angles)

  Обертає камеру по орбіті навколо центру коодинат.

Приватні дані

- mat4 rotation

  Матриця лінійних перетворень у якій зберігається обернення камери.
- vec4 position

  Вектор позіції камери $(x, y, z, 1.0)$.
- vec4 animation

  Вектор, який реалізования для інтерфейсу Bijective

7.4.1 Детальний опис

Клас камери, якою можна маніпулювати у сцені.

Див. визначення в файлі camera.h, рядок 19

7.4.2 Конструктор(и)

7.4.2.1 Camera() Camera::Camera ( )

Див. визначення в файлі camera.cpp, рядок 6

7.4.3 Опис методів компонент

7.4.3.1 add_animation() void Camera::add_animation (vec3 angles, float delta_time) [override], [virtual]

Метод реалізований для інтерфейсу Bijective

Переозначення з Bijective.

Див. визначення в файлі camera.cpp, рядок 69

7.4.3.2 get_animation() void Camera::get_animation (vec3 animation) [override], [virtual]

Метод реалізований для інтерфейсу Bijective

Переозначення з Bijective.

Див. визначення в файлі camera.cpp, рядок 31

7.4.3.3 get_direction() void Camera::get_direction (vec4 dest)

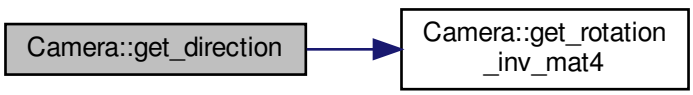
Обчислює напрямок перегляду камери.

Аргументи

out	dest	напрямок перегляду камери

Див. визначення в файлі camera.cpp, рядок 77

Граф всіх викликів цієї функції:



7.4.3.4 get_mat4() void Camera::get_mat4 (mat4 matrix) [override], [virtual]

Забирає матрицю лінійних перетворень над камерою.

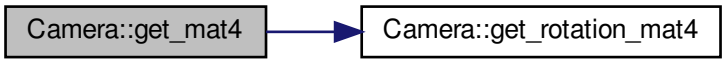
Аргументи

out	matrix	матриця лінійних перетворень

Переозначення з Bijective.

Див. визначення в файлі camera.cpp, рядок 41

Граф всіх викликів цієї функції:



7.4.3.5 get_position() void Camera::get_position (vec4 position) [override], [virtual]

Забирає поточну позицію камери у просторі.

Аргументи

out	position	поточна позицію камери

Переозначення з Bijective.

7.4.3.6 get_rotation() void Camera::get_rotation (vec3 rotation) [override], [virtual]

Забирає поточний нахил камери.

Аргументи

out	rotation	поточний нахил камери

Переозначення з Bijective.

Див. визначення в файлі camera.cpp, рядок 21

7.4.3.7 get_rotation_inv_mat4() void Camera::get_rotation_inv_mat4 (mat4 dest)

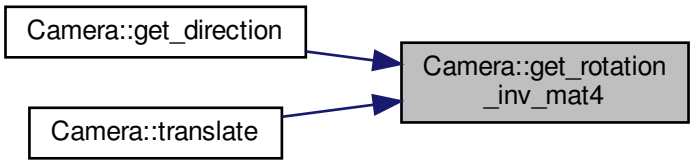
Забирає інвертовану матрицю обертання камери.

Аргументи

out	dest	інвертована матриця обертання камери.

Див. визначення в файлі camera.cpp, рядок 88

Граф викликів для цієї функції:



7.4.3.8 get_rotation_mat4() void Camera::get_rotation_mat4 (mat4 dest)

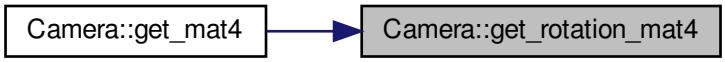
Забирає матрицю обертання камери.

Аргументи

out	dest	матриця обертання камери.

Див. визначення в файлі camera.cpp, рядок 83

Граф викликів для цієї функції:



7.4.3.9 rotate() void Camera::rotate (enum coord_enum coord, float delta_time) [override], [virtual]

Обертає камеру.

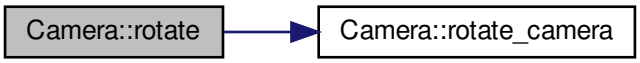
Аргументи

in	coord	позначає координатну вісь навколо якої обертати
in	delta_time	скільки часу пройшло з останнього кадру

Переозначення з Bijective.

Див. визначення в файлі camera.cpp, рядок 58

Граф всіх викликів цієї функції:



7.4.3.10 rotate_all() void Camera::rotate_all (vec3 angles) [override], [virtual]

Обернути камеру по всім осям.

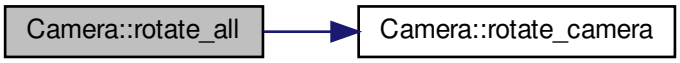
Аргументи

in	angles	вектор кутів у радіанах на кожну вісь

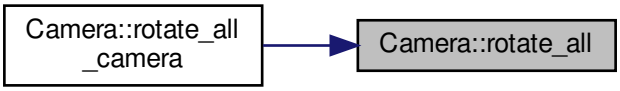
Переозначення з Bijective.

Див. визначення в файлі camera.cpp, рядок 63

Граф всіх викликів цієї функції:



Граф викликів для цієї функції:



7.4.3.11 rotate_all_camera() void Camera::rotate_all_camera (vec3 angles)

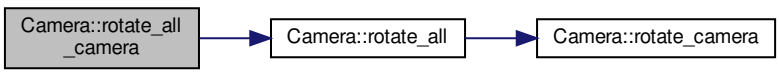
Обертає камеру по всім осям незалежно від часу.

Аргументи

angles	вектор кутів обертання по трьом осям (x, y, z)

Див. визначення в файлі camera.cpp, рядок 110

Граф всіх викликів цієї функції:



7.4.3.12 rotate_all_inverse() void Camera::rotate_all_inverse (vec3 angles)

Обертає камеру по орбіті навколо центру коодинат.

Аргументи

angles	вектор кутів у радіанах обертання по трьом осям (x, y, z)

Див. визначення в файлі camera.cpp, рядок 114

7.4.3.13 rotate_camera() void Camera::rotate_camera (float angle, enum coord_enum coord)

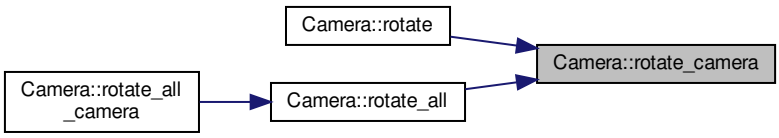
Обертає камеру по осі незалежно від часу.

Аргументи

angle	кут обертання у радіанах
coord	ввісь обертання

Див. визначення в файлі camera.cpp, рядок 96

Граф викликів для цієї функції:



7.4.3.14 set_animation() void Camera::set_animation (vec3 animation) [override], [virtual]

Метод реалізований для інтерфейсу Bijective

Переозначення з Bijective.

Див. визначення в файлі camera.cpp, рядок 36

7.4.3.15 set_position() void Camera::set_position (vec4 position) [override], [virtual]

Задає нову позицію камери у просторі.

Аргументи

in	position	нова позиція камери у просторі

Переозначення з Bijective.

Див. визначення в файлі camera.cpp, рядок 16

7.4.3.16 set_rotation() void Camera::set_rotation (vec3 rotation) [override], [virtual]

Задає новий нахил камери.

Аргументи

in	rotation	новий нахил камери

Переозначення з Bijective.

Див. визначення в файлі camera.cpp, рядок 26

7.4.3.17 translate() void Camera::translate (vec3 distances, float delta_time) [override], [virtual]

Переміщує камеру у просторі.

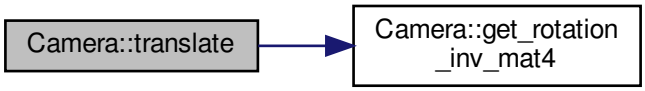
Аргументи

in	distances	відстані переміщення по осям
in	delta_time	скільки часу пройшло з останнього кадру

Переозначення з Bijective.

Див. визначення в файлі camera.cpp, рядок 46

Граф всіх викликів цієї функції:



7.4.3.18 translate_camera() void Camera::translate_camera (vec3 distances)

Переміщує камеру незалежно від часу.

Аргументи

distances	відстані переміщення по трьом осям (x, y, z)

Див. визначення в файлі camera.cpp, рядок 92

7.4.4 Компонентні дані

7.4.4.1 animation vec4 Camera::animation [private]

Вектор, який реалізования для інтерфейсу Bijective

Див. визначення в файлі camera.h, рядок 31

7.4.4.2 position vec4 Camera::position [private]

Вектор позіції камери $(x, y, z, 1.0)$.

Див. визначення в файлі camera.h, рядок 27

7.4.4.3 rotation mat4 Camera::rotation [private]

Матриця лінійних перетворень у якій зберігається обернення камери.

Див. визначення в файлі camera.h, рядок 23

Документація цих класів була створена з файлів:

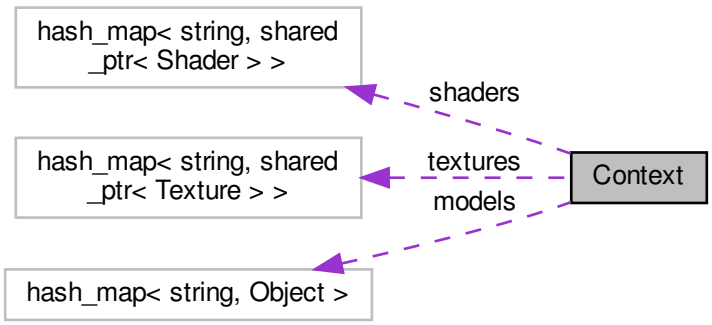
- osdo/camera.h
- osdo/camera.cpp

7.5 Структура Context

Контекст, який зберігає усі завантажені у пам'ять ресурси.

#include <context.h>

Діаграма зв'язків класу Context:



Загальнодоступні типи

- typedef hash_map< string, Object > Models
  Тип для зберігання моделей.
- typedef hash_map< string, shared_ptr< Texture > > Textures
  Тип для зберігання текстур.

Загальнодоступні елементи

- Context ()
- Models::iterator & next_active ()
  перехід до наступної моделі для редагування.
- void load_texture (const char *path)
  Завантажує текстуру у пам'ять.
- bool load_shader (const char *name, const Shader::shader_map &shaders)
  Завантажує та компілює шейдер.
- bool load_model (const string &path)
  Завантажує модель з поверхнями Безье

Загальнодоступні атрибути

- Models models
  Завантажені моделі.
- hash_map< string, shared_ptr< Shader > > shaders
  Зкомпіловані шейдери.
- Textures textures
  Завантажені текстури.
- Models::iterator active
  Вибрана модель для редагування.
- Textures::iterator active_texture
  Вибрана текстура для відображення.

7.5.1 Детальний опис

Контекст, який зберігає усі завантажені у пам'ять ресурси.

Див. визначення в файлі context.h, рядок 26

7.5.2 Опис типів користувача

7.5.2.1 Models typedef hash_map<string, Object> Context::Models

Тип для зберігання моделей.

Див. визначення в файлі context.h, рядок 31

7.5.2.2 Textures typedef hash_map<string, shared_ptr<Texture> > Context::Textures

Тип для зберігання текстур.

Див. визначення в файлі context.h, рядок 35

7.5.3 Конструктор(и)

7.5.3.1 Context() Context::Context ( )

Див. визначення в файлі context.cpp, рядок 6

7.5.4 Опис методів компонент

7.5.4.1 load_model() bool Context::load_model (const string & path)

Завантажує модель з поверхнями Безье

Аргументи

path	шлях до файлу з моделлю

Повертає

статус успішності завантаження моделі

7.5.4.2 load_shader()

bool Context::load_shader (
const char * name,
const Shader::shader_map & shaders)

Завантажує та компілює шейдер.

Аргументи

name	назва шейдеру
shaders	массив до файлів шейдеру

Повертає

статус успішності завантаження та компіляції шейдеру

Див. визначення в файлі context.cpp, рядок 26

Граф всіх викликів цієї функції:

7.5.4.3 load_texture()

void Context::load_texture (
const char * path)

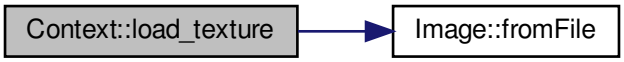
Завантажує текстуру у пам'ять.

Аргументи

in	path	шлях до файлу з текстурою

Див. визначення в файлі context.cpp, рядок 17

Граф всіх викликів цієї функції:



7.5.4.4 next_active() Context::Models::iterator & Context::next_active ( )

перехід до наступної моделі для редагування.

Повертає

ітератор моделі

Див. визначення в файлі context.cpp, рядок 10

7.5.5 Компонентні дані

7.5.5.1 active Models::iterator Context::active

Вибрана модель для редагування.

Див. визначення в файлі context.h, рядок 52

7.5.5.2 active_texture Textures::iterator Context::active_texture

Вибрана текстура для відображення.

Див. визначення в файлі context.h, рядок 56

7.5.5.3 models Models Context::models

Завантажені моделі.

Див. визначення в файлі context.h, рядок 39

7.5.5.4 shaders hash_map<string, shared_ptr<Shader> > Context::shaders

Зкомпіловані шейдери.

Див. визначення в файлі context.h, рядок 43

7.5.5.5 textures Textures Context::textures

Завантажені текстури.

Див. визначення в файлі context.h, рядок 47

Документація цих структур була створена з файлів:

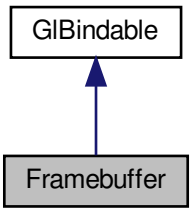
- osdo/context.h
- osdo/context.cpp

7.6 Клас Framebuffer

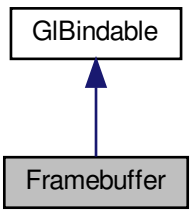
Буфер кадру, що використовується для рендеренгу.

#include <framebuffer.h>

Схема успадкувань для Framebuffer



Діаграма зв'язків класу Framebuffer:



Загальнодоступні елементи

- Framebuffer ()
- ~Framebuffer () override
- bool check (GLenum target=GL_FRAMEBUFFER)

  Перевіряє, чи готовий буфер до рендеренгу.
- void tex_2d_multisample (GLsizei size[2], const Texture &texture)

  Підготовлює текстуру з згладжуванням
- void tex_2d (GLsizei size[2], const Texture &texture)

  Підготовлює звичайну текстуру
- void renderbuffer (const Renderbuffer &rb, GLenum target) const

  Підготовлює рендер буфер rb
- void rb_color_multisample (GLsizei size[2], const Renderbuffer &rb) const

  Підготовлює рендер буфер з згладжуванням для зберігання кольорів.
- void rb_depth_multisample (GLsizei size[2], const Renderbuffer &rb) const

  Підготовлює глибинний рендер буфер з згладжуванням.
- void rb_color (GLsizei size[2], const Renderbuffer &rb) const

  Підготовлює рендер буфер для зберігання кольорів.
- void rb_depth (GLsizei size[2], const Renderbuffer &rb) const

  Підготовлює глибинний рендер буфер.

Приватні елементи

- GLuint _generate () const override

  Віртуальний метод що створює буфер кадру.
- virtual void _bind (const GLuint id, GLenum target) const override

  Віртуальний метод, що прив'язує контекст до буферу кадру.
- virtual GLenum _default () const override

  Віртуальний метод, що задає ціль прив'язки за замовчуванням.

Додаткові успадковані елементи

7.6.1 Детальний опис

Буфер кадру, що використовується для рендеренгу.

Див. визначення в файлі framebuffer.h, рядок 17

7.6.2 Конструктор(и)

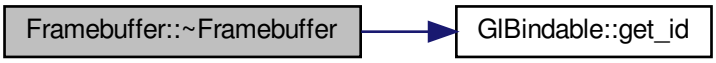
7.6.2.1 Framebuffer() Framebuffer::Framebuffer ( )

Див. визначення в файлі framebuffer.cpp, рядок 22

7.6.2.2 ~Framebuffer() Framebuffer::~Framebuffer ( ) [override]

Див. визначення в файлі framebuffer.cpp, рядок 24

Граф всіх викликів цієї функції:



7.6.3 Опис методів компонент

7.6.3.1 _bind() void Framebuffer::_bind (
const GLuint id,
GLenum target) const [override], [private], [virtual]

Віртуальний метод, що прив'язує контекст до буферу кадру.

Аргументи

id	індекс буфера кадру
target	ціль буфера кадру

Переозначення з GlBindable.

Див. визначення в файлі framebuffer.cpp, рядок 12

7.6.3.2 _default() GLenum Framebuffer::_default ( ) const [override], [private], [virtual]

Віртуальний метод, що задає ціль прив'язки за замовчуванням.

Повертає

тип ціль прив'язки за замовчуванням

Переозначення з GlBindable.

Див. визначення в файлі framebuffer.cpp, рядок 17

7.6.3.3 _generate() GLuint Framebuffer::_generate ( ) const [override], [private], [virtual]

Віртуальний метод що створює буфер кадру.

Повертає

індекс буфера кадру

Переозначення з GlBindable.

Див. визначення в файлі framebuffer.cpp, рядок 5

7.6.3.4 check() bool Framebuffer::check (GLenum target = GL_FRAMEBUFFER)

Перевіряє, чи готовий буфер до рендеренгу.

Аргументи

target	тип буфера

Повертає

статус буфера

Див. визначення в файлі framebuffer.cpp, рядок 28

Граф викликів для цієї функції:

7.6.3.5 rb_color() void Framebuffer::rb_color (GLsizei size[2], const Renderbuffer & rb) const

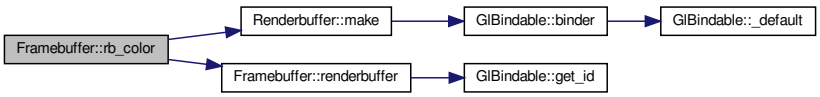
Підготовлює рендер буфер для зберігання кольорів.

Аргументи

size	ширина та висота кадру
rb	рендер буфер

Див. визначення в файлі framebuffer.cpp, рядок 61

Граф всіх викликів цієї функції:



7.6.3.6 rb_color_multisample() void Framebuffer::rb_color_multisample (GLsizei size[2], const Renderbuffer & rb) const

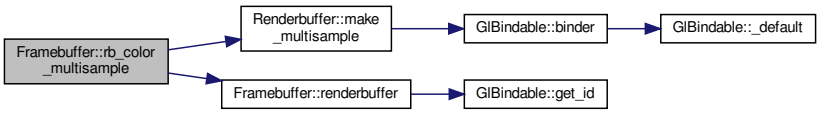
Підготовлює рендер буфер з згладжуванням для зберігання кольорів.

Аргументи

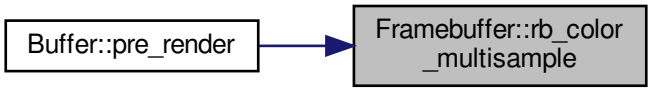
size	ширина та висота кадру
rb	рендер буфер

Див. визначення в файлі framebuffer.cpp, рядок 49

Граф всіх викликів цієї функції:



Граф викликів для цієї функції:



7.6.3.7 rb_depth() void Framebuffer::rb_depth (GLsizei size[2], const Renderbuffer & rb) const

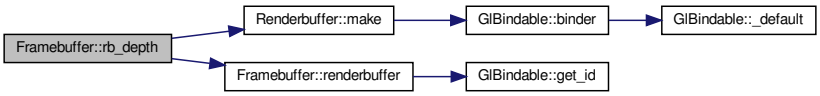
Підготовлює глибинний рендер буфер.

Аргументи

size	ширина та висота кадру
rb	рендер буфер

Див. визначення в файлі framebuffer.cpp, рядок 67

Граф всіх викликів цієї функції:



7.6.3.8 rb_depth_multisample() void Framebuffer::rb_depth_multisample (GLsizei size[2], const Renderbuffer & rb) const

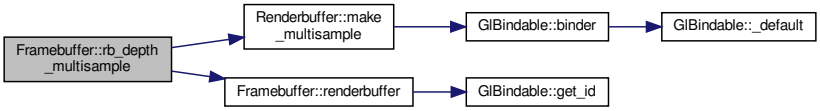
Підготовлює глибинний рендер буфер з згладжуванням.

Аргументи

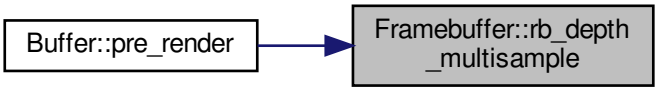
size	ширина та висота кадру
rb	рендер буфер

Див. визначення в файлі framebuffer.cpp, рядок 55

Граф всіх викликів цієї функції:



Граф викликів для цієї функції:



7.6.3.9 renderbuffer() void Framebuffer::renderbuffer (const Renderbuffer & rb, GLenum target) const

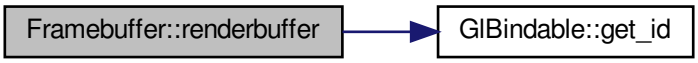
Підготовлює рендер буфер rb

Аргументи

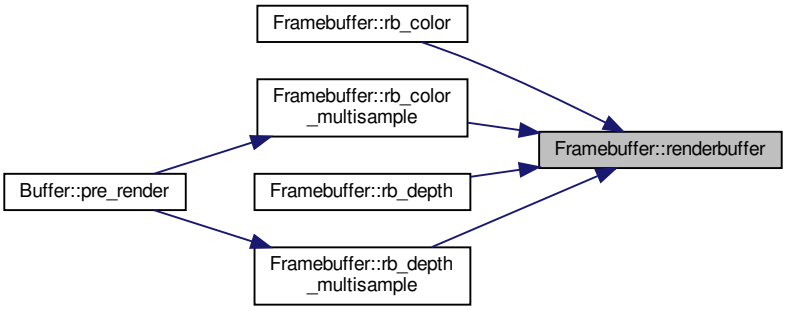
rb	рендер буфер
target	тип буфера

Див. визначення в файлі framebuffer.cpp, рядок 44

Граф всіх викликів цієї функції:



Граф викликів для цієї функції:



7.6.3.10 tex_2d()

void Framebuffer::tex_2d (
GLsizei size[2],
const Texture & texture)

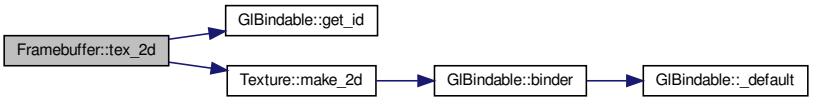
Підготовлює звичайну текстуру

Аргументи

size	ширина та висота кадру
texture	текстура у яку зберігають кадр

Див. визначення в файлі framebuffer.cpp, рядок 38

Граф всіх викликів цієї функції:



Граф викликів для цієї функції:

7.6.3.11 tex_2d_multisample() void Framebuffer::tex_2d_multisample (GLsizei size[2], const Texture & texture)

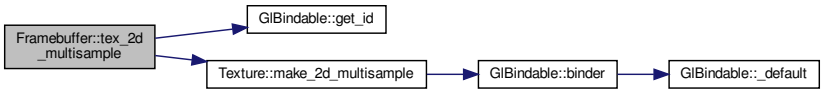
Підготовлює текстуру з згладжуванням

Аргументи

size	ширина та висота кадру
texture	текстура у яку зберігають кадр з згладжуванням

Див. визначення в файлі framebuffer.cpp, рядок 32

Граф всіх викликів цієї функції:



Документація цих класів була створена з файлів:

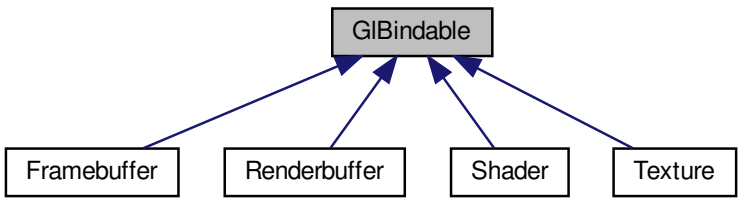
- osdo/framebuffer.h
- osdo/framebuffer.cpp

7.7 Клас GlBindable

Абстрактний клас, який виконує роль генерації та прив'язки об'єктів OpenGL.

#include <glbindable.h>

Схема успадкувань для GlBindable



Загальнодоступні елементи

- virtual ~GlBindable ()
- GlBindable (const GlBindable &)=delete
- GlBindable (GlBindable &&)=delete
- GlBindable & operator= (const GlBindable &)=delete
- GlBindable & operator= (GlBindable &&)=delete
- const GLuint & get_id () const

  Повертає індекс об'єкту OpenGL.
- void * get_vid () const

  Повертає індекс об'єкту OpenGL у типі void*
- void bind () const

  Прив'язує контекст до об'єкту із ціллю за замовчуванням.
- void bind (GLenum target) const

  Прив'язує контекст до об'єкту із ціллю target
- void unbind () const

  Відв'язує контекст до об'єкту із ціллю за замовчуванням.
- void unbind (GLenum target) const

  Відв'язує контекст до об'єкту із ціллю target
- GlBinder binder () const

  Повертає об'єкт прив'язки контексту із ціллю за замовчуванням
- GlBinder binder (GLenum target) const

  Повертає об'єкт прив'язки контексту із ціллю target

Захищені елементи

- virtual GLuint _generate () const

  Віртуальний метод що створює об'єкт.
- virtual void _bind (const GLuint id, GLenum target) const

  Віртуальний метод, що прив'язує контекст OpenGL до об'єкту.
- virtual GLenum _default () const

  Віртуальний метод, що задає ціль прив'язки за замовчуванням.
- GlBindable ()
- GlBindable (const GLuint id)

  Конструктор з параметром.

Приватні дані

- const GLuint id

  Індекс об'єкту OpenGL.

7.7.1 Детальний опис

Абстрактний клас, який виконує роль генерації та прив'язки об'єктів OpenGL.

Див. визначення в файлі glbindable.h, рядок 15

7.7.2 Конструктор(и)

7.7.2.1 GlBindable() [1/4] GlBindable::GlBindable ( ) [protected]

Див. визначення в файлі glbindable.cpp, рядок 14

7.7.2.2 GlBindable() [2/4] GlBindable::GlBindable (const GLuint id) [protected]

Конструктор з параметром.

Аргументи

id	індекс об'єкту OpenGL

Див. визначення в файлі glbindable.cpp, рядок 16

7.7.2.3 ~GlBindable() GlBindable::~GlBindable ( ) [virtual]

Див. визначення в файлі glbindable.cpp, рядок 18

7.7.2.4 GlBindable() [3/4] GlBindable::GlBindable (const GlBindable &) [delete]

7.7.2.5 GlBindable() [4/4] GlBindable::GlBindable (GlBindable &&) [delete]

7.7.3 Опис методів компонент

7.7.3.1 _bind() void GlBindable::_bind (const GLuint id, GLenum target) const [protected], [virtual]

Віртуальний метод, що прив'язує контекст OpenGL до об'єкту.

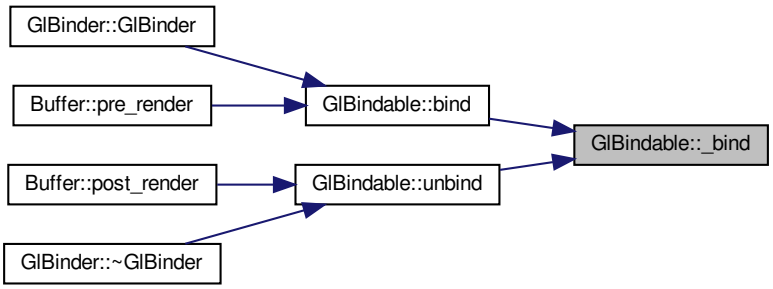
Аргументи

id	індекс об'єкту OpenGL
target	ціль прив'язки об'єкту

Переозначається в Texture, Shader, Renderbuffer і Framebuffer.

Див. визначення в файлі glbindable.cpp, рядок 8

Граф викликів для цієї функції:



7.7.3.2 _default() GLenum GlBindable::_default ( ) const [protected], [virtual]

Віртуальний метод, що задає ціль прив'язки за замовчуванням.

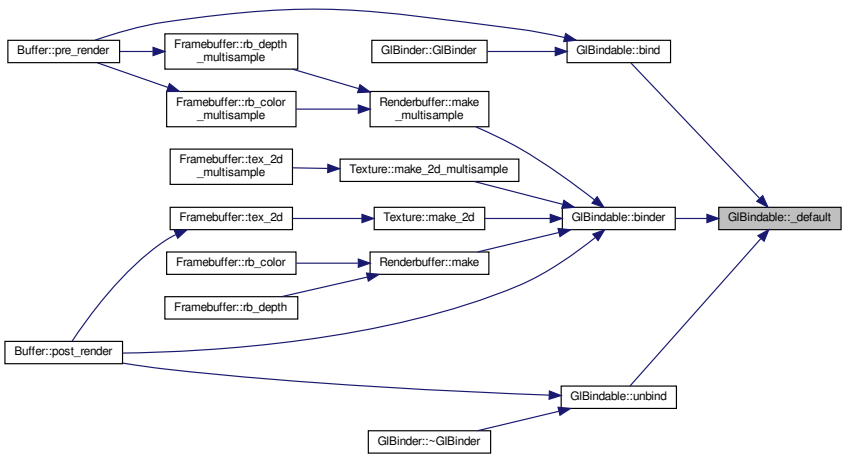
Повертає

ціль прив'язки за замовчуванням

Переозначається в Texture, Renderbuffer і Framebuffer.

Див. визначення в файлі glbindable.cpp, рядок 10

Граф викликів для цієї функції:



7.7.3.3 _generate() GLuint GlBindable::_generate ( ) const [protected], [virtual]

Віртуальний метод що створює об'єкт.

Повертає

індекс об'єкту OpenGL

Переозначається в Texture, Renderbuffer і Framebuffer.

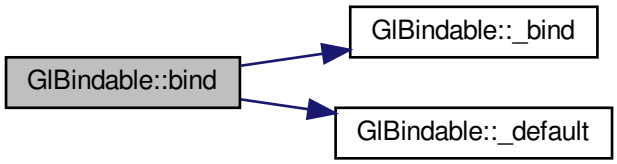
Див. визначення в файлі glbindable.cpp, рядок 4

7.7.3.4 bind() [1/2] void GlBindable::bind ( ) const

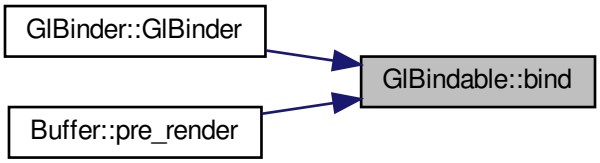
Прив'язує контекст до об'єкту із ціллю за замовчуванням.

Див. визначення в файлі glbindable.cpp, рядок 30

Граф всіх викликів цієї функції:



Граф викликів для цієї функції:



7.7.3.5 bind() [2/2] void GlBindable::bind (GLenum target) const

Прив'язує контекст до об'єкту із ціллю target

Аргументи

target	ціль прив'язки об'єкту

Див. визначення в файлі glbindable.cpp, рядок 35

Граф всіх викликів цієї функції:

7.7.3.6 binder() [1/2] GlBinder GlBindable::binder ( ) const

Повертає об'єкт прив'язки контексту із ціллю за замовчуванням

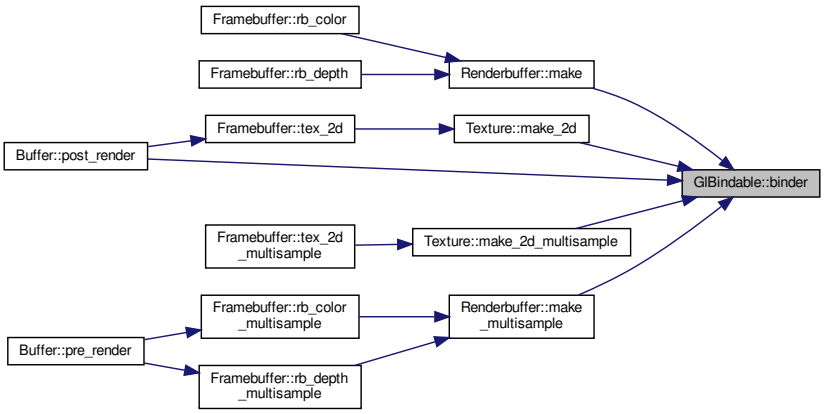
Повертає

об'єкт прив'язки контексту із ціллю за замовчуванням

Див. визначення в файлі glbindable.cpp, рядок 50

Граф всіх викликів цієї функції:

Граф викликів для цієї функції:



7.7.3.7 binder() [2/2] GlBinder GlBindable::binder (GLenum target) const

Повертає об'єкт прив'язки контексту із ціллю target

Аргументи

target	ціль прив'язки об'єкту

Повертає

об'єкт прив'язки контексту із ціллю target

Див. визначення в файлі glbindable.cpp, рядок 55

7.7.3.8 get_id() const GLuint & GlBindable::get_id ( ) const

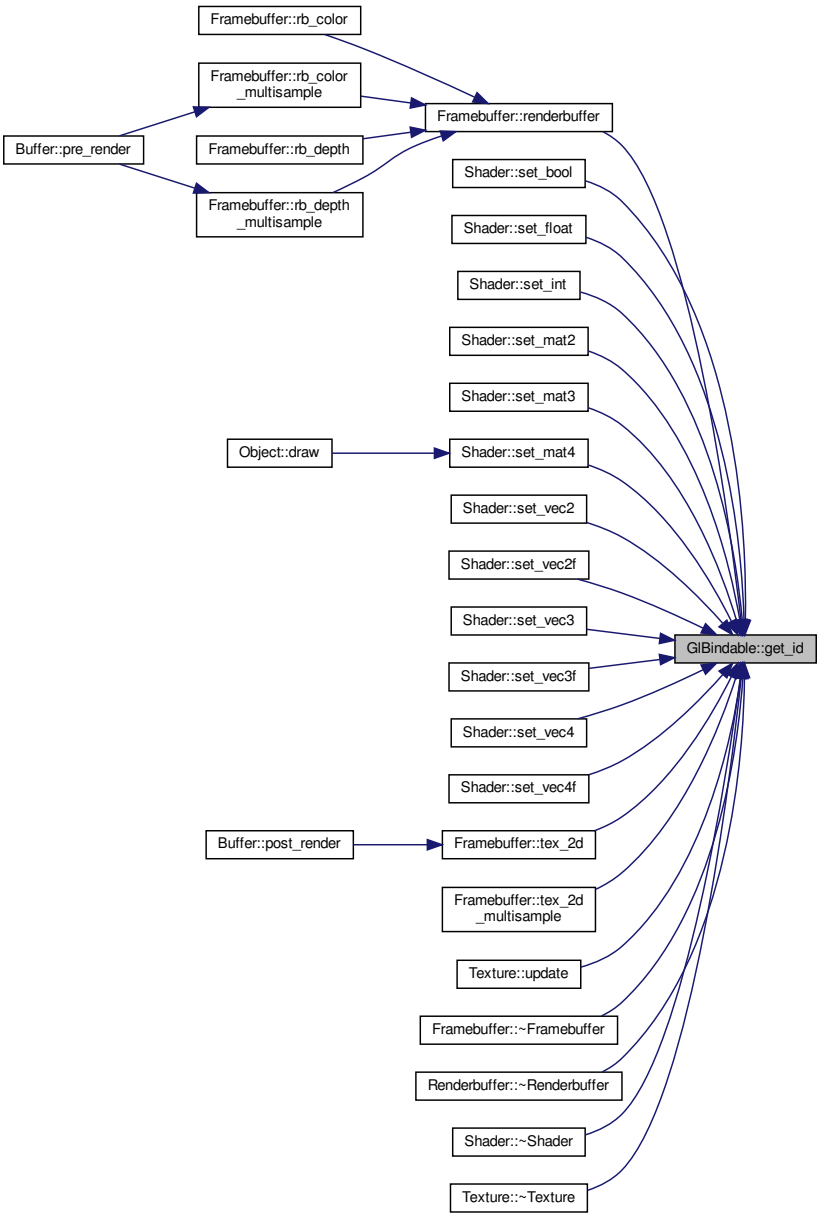
Повертає індекс об'єкту OpenGL.

Повертає

індекс об'єкту OpenGL

Див. визначення в файлі glbindable.cpp, рядок 20

Граф викликів для цієї функції:



7.7.3.9 get_vid() void * GlBindable::get_vid ( ) const

Повертає індекс об'єкту OpenGL у типі void*

Повертає

індекс об'єкту OpenGL

Див. визначення в файлі glbindable.cpp, рядок 25

7.7.3.10 operator=() [1/2] GlBindable& GlBindable::operator= (const GlBindable &) [delete]

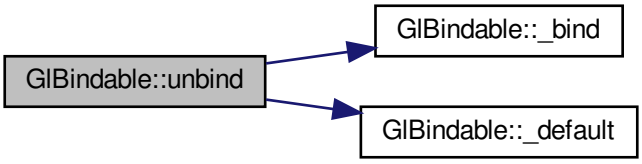
7.7.3.11 operator=() [2/2] GlBindable& GlBindable::operator= (GlBindable &&) [delete]

7.7.3.12 unbind() [1/2] void GlBindable::unbind ( ) const

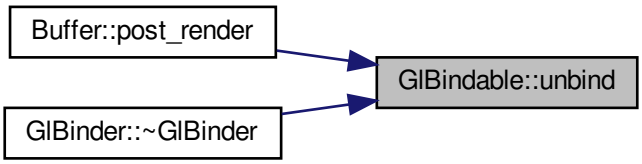
Відв'язує контекст до об'єкту із ціллю за замовчуванням.

Див. визначення в файлі glbindable.cpp, рядок 40

Граф всіх викликів цієї функції:



Граф викликів для цієї функції:



7.7.3.13 unbind() [2/2] void GlBindable::unbind (GLenum target) const

Відв'язує контекст до об'єкту із ціллю target

Аргументи

target	ціль прив'язки об'єкту

Див. визначення в файлі glbindable.cpp, рядок 45

Граф всіх викликів цієї функції:

7.7.4 Компонентні дані

7.7.4.1 id const GLuint GlBindable::id [private]

Індекс об'єкту OpenGL.

Див. визначення в файлі glbindable.h, рядок 20

Документація цих класів була створена з файлів:

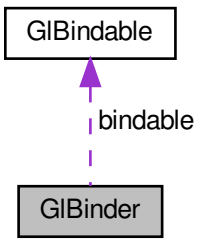
- osdo/glbindable.h
- osdo/glbindable.cpp

7.8 Клас GlBinder

Клас який прив'язує контексту до деякого об'єкту OpenGL.

#include <glbinder.h>

Діаграма зв'язків класу GlBinder:



Загальнодоступні елементи

- GlBinder (const GlBindable &bindable, GLenum target)
  Конструктор із параметром
- ~GlBinder ()

Приватні дані

- const GlBindable & bindable
  Об'єкт OpenGL, який прив'язують.
- const GLenum target
  Ціль прив'язки.

7.8.1 Детальний опис

Клас який прив'язує контексту до деякого об'єкту OpenGL.

Див. визначення в файлі glbinder.h, рядок 15

7.8.2 Конструктор(и)

7.8.2.1 GlBinder() GlBinder::GlBinder (const GlBindable & bindable, GLenum target)

Конструктор із параметром

Аргументи

bindable	об'єкт OpenGL, який прив'язують
target	ціль прив'язки

Див. визначення в файлі glbinder.cpp, рядок 5

Граф всіх викликів цієї функції:

7.8.2.2 ~GlBinder() GlBinder::~GlBinder ( )

Див. визначення в файлі glbinder.cpp, рядок 10

Граф всіх викликів цієї функції:

7.8.3 Компонентні дані

7.8.3.1 bindable const GlBindable& GlBinder::bindable [private]

Об'єкт OpenGL, який прив'язують.

Див. визначення в файлі glbinder.h, рядок 20

7.8.3.2 target const GLenum GlBinder::target [private]

Ціль прив'язки.

Див. визначення в файлі glbinder.h, рядок 24

Документація цих класів була створена з файлів:

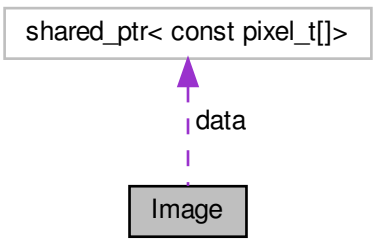
- osdo/glbinder.h
- osdo/glbinder.cpp

7.9 Клас Image

Зберігає масив пікселів, ширину та висоту.

#include <image.h>

Діаграма зв'язків класу Image:



Загальнодоступні елементи

- Image (shared_ptr< const pixel_t[]> data, const int width, const int height)

  Конструктор для зображення. Зберігає посилання на масив пікселів, а також висоту та ширину зображення.

Загальнодоступні статичні елементи

- static Image fromFile (const char *path)

  Зчитування зображення з файлу.

Загальнодоступні атрибути

- shared_ptr< const pixel_t[]> data

  Константнний масив пікселів. Зберігає розумний вказівник на масив пікселів зображення розміром висота*ширина.
- const int width

  Ширина зображення.
- const int height

  Висота зображення.

7.9.1 Детальний опис

Зберігає масив пікселів, ширину та висоту.

Див. визначення в файлі image.h, рядок 36

7.9.2 Конструктор(и)

7.9.2.1 Image()

```
Image::Image (
        shared_ptr< const pixel_t[]> data,
        const int width,
        const int height )
```

Конструктор для зображення. Зберігає посилання на масив пікселів, а також висоту та ширину зображення.

Аргументи

data	Розумне посилання на масив пікселів, розмір повинен бути висота * ширина.
width	Ширина зображення.
height	Висота зображення.

Див. визначення в файлі image.cpp, рядок 9

7.9.3 Опис методів компонент

7.9.3.1 fromFile() Image Image::fromFile (const char * path) [static]

Зчитування зображення з файлу.

Аргументи

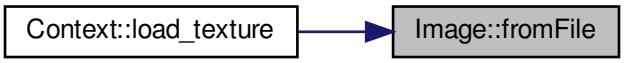
path	Шлях до файлу зображення.

Повертає

клас Image, який посилається на пікселі.

Див. визначення в файлі image.cpp, рядок 15

Граф викликів для цієї функції:



7.9.4 Компонентні дані

7.9.4.1 data shared_ptr<const pixel_t[]> Image::data

Константнний масив пікселів. Зберігає розумний вказівник на масив пікселів зображення розміром висота*ширина.

Див. визначення в файлі image.h, рядок 44

7.9.4.2 height const int Image::height

Висота зображення.

Див. визначення в файлі image.h, рядок 52

7.9.4.3 width const int Image::width

Ширина зображення.

Див. визначення в файлі image.h, рядок 48

Документація цих класів була створена з файлів:

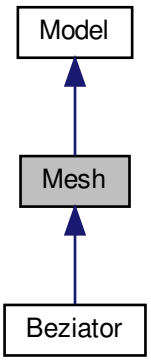
- osdo/image.h
- osdo/image.cpp

7.10 Клас Mesh

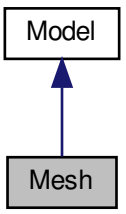
Меш, який зберігається на відеокарті.

#include <mesh.h>

Схема успадкувань для Mesh



Діаграма зв'язків класу Mesh:



Загальнодоступні елементи

- Mesh ()
- ~Mesh () override
- Mesh (const Mesh &)=delete
- Mesh (Mesh &&)=delete
- Mesh & operator= (const Mesh &)=delete
- Mesh & operator= (Mesh &&)=delete
- void cube_update ()

  Завантажити у відеокарту меш куба.
- void update (const Vertex *vertices, size_t vertices_n, const GLuint *indices, size_t indices_n)

  Завантажити у відеокарту новий меш.
- void draw (Shader &shader, bool pre_generated) override

  Відображує меш.
- void draw_mode (GLenum mode)

  Відображує меш у певному режимі. Див. glDrawElements.

Захищені дані

- GLuint vao

  Об'єкт масиву вершин. Розшифровується "Vertex Array Object".
- GLuint vbo

  Об'єкт буфера вершин. Розшифровується "Vertex Buffer Object".
- GLuint ebo

  Об'єкти буфера елементів. Розшифровується "Element Buffer Objects".
- GLint indices_size

  Кількість індексів у ebo.

7.10.1 Детальний опис

Меш, який зберігається на відеокарті.

Див. визначення в файлі mesh.h, рядок 15

7.10.2 Конструктор(и)

7.10.2.1 Mesh() [1/3] Mesh::Mesh ( )

Див. визначення в файлі mesh.cpp, рядок 5

7.10.2.2 ~Mesh() Mesh::~Mesh ( ) [override]

Див. визначення в файлі mesh.cpp, рядок 40

7.10.2.3 Mesh() [2/3] Mesh::Mesh (const Mesh &) [delete]

7.10.2.4 Mesh() [3/3] Mesh::Mesh (Mesh &&) [delete]

7.10.3 Опис методів компонент

7.10.3.1 cube_update() void Mesh::cube_update ( )

Завантажити у відеокарту меш куба.

Див. визначення в файлі mesh.cpp, рядок 46

7.10.3.2 draw() void Mesh::draw (Shader & shader, bool pre_generated) [override], [virtual]

Відображує меш.

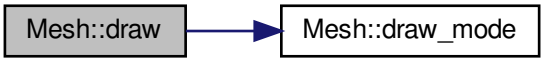
Аргументи

shader	Шейдер який використовуєтсья для відображення мешу.
pre_generated	флаг залишений для інтерфесу, але не використовується.

Переозначення з Model.

Див. визначення в файлі mesh.cpp, рядок 87

Граф всіх викликів цієї функції:



7.10.3.3 draw_mode() void Mesh::draw_mode (GLenum mode)

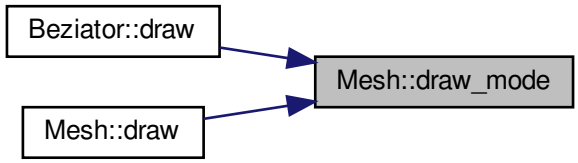
Відображує меш у певному режимі. Див. glDrawElements.

Аргументи

mode	режим відображення.

Див. визначення в файлі mesh.cpp, рядок 73

Граф викликів для цієї функції:



7.10.3.4 operator=() [1/2] Mesh& Mesh::operator= (const Mesh &) [delete]

7.10.3.5 operator=() [2/2] Mesh& Mesh::operator= (Mesh &&) [delete]

7.10.3.6 update() void Mesh::update (const Vertex * vertices, size_t vertices_n, const GLuint * indices, size_t indices_n)

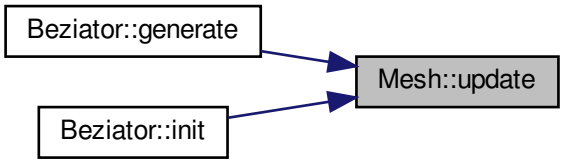
Завантажити у відеокарту новий меш.

Аргументи

vertices	масив вершин
vertices_n	кількість вершин
indices	масив індексів вершин
indices_n	кількість індексів вершин

Див. визначення в файлі mesh.cpp, рядок 53

Граф викликів для цієї функції:



7.10.4 Компонентні дані

7.10.4.1 ebo GLuint Mesh::ebo [protected]

Об'єкти буфера елементів. Розшифровується "Element Buffer Objects".

Див. визначення в файлі mesh.h, рядок 28

7.10.4.2 indices_size GLint Mesh::indices_size [protected]

Кількість індексів у ebo.

Див. визначення в файлі mesh.h, рядок 32

7.10.4.3 vao GLuint Mesh::vao [protected]

Об'єкт масиву вершин. Розшифровується "Vertex Array Object".

Див. визначення в файлі mesh.h, рядок 20

7.10.4.4 vbo GLuint Mesh::vbo [protected]

Об'єкт буфера вершин. Розшифровується "Vertex Buffer Object".

Див. визначення в файлі mesh.h, рядок 24

Документація цих класів була створена з файлів:

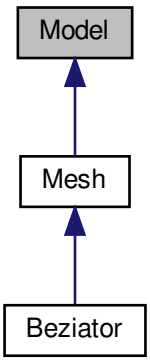
- osdo/mesh.h
- osdo/mesh.cpp

7.11 Клас Model

Інтерфейс до деякої моделі, яку можна відобразити.

#include <model.h>

Схема успадкувань для Model



Загальнодоступні елементи

- virtual ~Model ()
- virtual void draw (Shader &shader, bool pre_generated=false)

  Відображує модель.
- virtual void generate (size_t d=8)

  Генерує деталізований меш моделі. Див. Beziator::generate
- virtual vector< Vertex > * get_vertices ()

  Видає список вершин моделі.
- virtual void edit_panel ()

  Створює вікно редагування моделі.

7.11.1 Детальний опис

Інтерфейс до деякої моделі, яку можна відобразити.

Див. визначення в файлі model.h, рядок 18

7.11.2 Конструктор(и)

7.11.2.1 ~Model() Model::~Model ( ) [virtual]

Див. визначення в файлі model.cpp, рядок 4

7.11.3 Опис методів компонент

7.11.3.1 draw() void Model::draw (Shader & shader, bool pre_generated = false) [virtual]

Відображує модель.

Аргументи

shader	Шейдер який використовуєтсья для відображення моделі.
pre_generated	флаг, який позначає яким чином відображати модель.

Переозначається в Mesh і Beziator.

Див. визначення в файлі model.cpp, рядок 6

7.11.3.2 edit_panel() void Model::edit_panel ( ) [virtual]

Створює вікно редагування моделі.

Див. визначення в файлі model.cpp, рядок 14

7.11.3.3 generate() void Model::generate (size_t d = 8) [virtual]

Генерує деталізований меш моделі. Див. Beziator::generate

Аргументи

d	ступінь деталізації.

Переозначається в Beziator.

Див. визначення в файлі model.cpp, рядок 8

7.11.3.4 get_vertices() vector< Vertex > * Model::get_vertices ( ) [virtual]

Видає список вершин моделі.

Повертає

Вказівник на поле vertices.

Переозначається в Beziator.

Див. визначення в файлі model.cpp, рядок 10

Документація цих класів була створена з файлів:

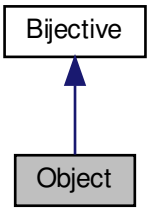
- osdo/model.h
- osdo/model.cpp

7.12 Клас Object

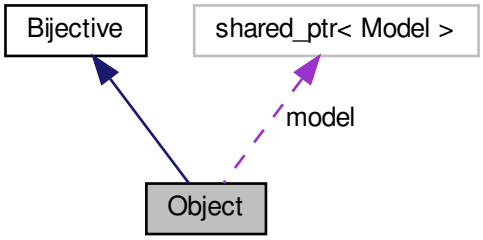
Об'єкт моделі.

#include <object.h>

Схема успадкувань для Object



Діаграма зв'язків класу Object:



Загальнодоступні елементи

- Object (shared_ptr< Model > model=nullptr)
  Конструктор, який створює об'єкт з моделі.
- ~Object () override=default
- void get_position (vec4 position) override
  Забирає поточну позицію об'єкта у просторі.
- void set_position (vec4 position) override
  Задає нову позицію об'єкта у просторі.
- void get_rotation (vec3 rotation) override
  Забирає поточний нахил об'єкта.
- void set_rotation (vec3 rotation) override
  Задає новий нахил об'єкта.
- void get_animation (vec3 rotation) override
  Забирає поточну анімацію обернення об'єкта.
- void set_animation (vec3 rotation) override
  Задає нову анімацію обернення об'єкта.
- void get_mat4 (mat4 matrix) override
  Забирає матрицю лінійних перетворень над об'єктом.
- void translate (vec3 distances, float delta_time) override
  Переміщує об'єкт у просторі.
- void rotate (enum coord_enum coord, float delta_time) override
  Обертає об'єкт.
- void rotate_all (vec3 angles) override
  Обернути об'єкт по всім осям.
- void add_animation (vec3 angles, float delta_time) override
  Додає швидкість анімації обертання об'єкту.
- shared_ptr< Model > get_model ()
  Повертає модель об'єкту.
- void draw (Shader &shader, mat4 mat4buf, GLdouble delta_time, bool pre_generated)
  Відображує об'єкт
- void translate_object (vec3 distances)
  Переміщує об'єкт незалежно від часу.
- void rotate_object (float angle, enum coord_enum coord)
  Обертає об'єкт по осі незалежно від часу.
- void rotate_all_object (vec3 angles)
  Обертає об'єкт по всім осям незалежно від часу.
- void animate (float step)
  Обертає об'єкт по заданій анімації.
- void scale (vec3 scale)
  Збільщує або зменшує об'єкт по трьом осям.
- mat4 * get_transform ()
  Повертає матрицю лінійних перетворень без переміщення.

Приватні дані

- mat4 transform
  Матриця лінійних перетворень.
- vec4 position
  Позиція об'єкту у просторі
- vec4 animation
  Анімація обертання по осям $(x, y, z, 1.0)$.
- shared_ptr< Model > model
  Модель об'єкту.

7.12.1 Детальний опис

Об'єкт моделі.

Див. визначення в файлі object.h, рядок 20

7.12.2 Конструктор(и)

7.12.2.1 Object() Object::Object (shared_ptr< Model > model = nullptr)

Конструктор, який створює об'єкт з моделі.

Аргументи

model	Модель об'єкту.

Див. визначення в файлі object.cpp, рядок 7

7.12.2.2 ~Object() Object::~Object ( ) [override], [default]

7.12.3 Опис методів компонент

7.12.3.1 add_animation() void Object::add_animation (vec3 angles, float delta_time) [override], [virtual]

Додає швидкість анімації обертання об'єкту.

Аргументи

in	angles	вектор швидкостей анімацій обертання по трьом осям
in	delta_time	скільки часу пройшло з останнього кадру

Переозначення з Bijective.

Див. визначення в файлі object.cpp, рядок 73

7.12.3.2 animate() void Object::animate (float step)

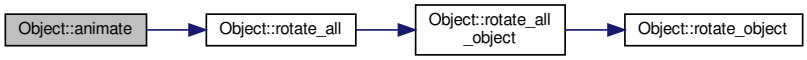
Обертає об'єкт по заданій анімації.

Аргументи

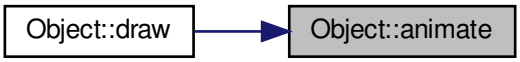
step	шаг анімації

Див. визначення в файлі object.cpp, рядок 105

Граф всіх викликів цієї функції:



Граф викликів для цієї функції:



7.12.3.3 draw() void Object::draw (Shader & shader, mat4 mat4buf, GLdouble delta_time, bool pre_generated)

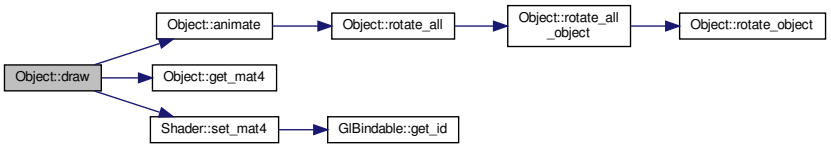
Відображує об'єкт

Аргументи

shader	Шейдер який використовуєтсья для відображення моделі
mat4buf	буфер матриці
delta_time	скільки часу пройшло з останнього кадру
pre_generated	флаг, який позначає яким чином відображати модель

Див. визначення в файлі object.cpp, рядок 56

Граф всіх викликів цієї функції:



7.12.3.4 get_animation() void Object::get_animation (vec3 rotation) [override], [virtual]

Забирає поточну анімацію обернення об'єкта.

Аргументи

out	rotation	поточна анімація обернення об'єкта

Переозначення з Bijective.

Див. визначення в файлі object.cpp, рядок 34

7.12.3.5 get_mat4() void Object::get_mat4 (mat4 matrix) [override], [virtual]

Забирає матрицю лінійних перетворень над об'єктом.

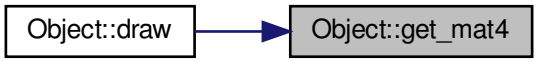
Аргументи

out	matrix	матриця лінійних перетворень

Переозначення з Bijective.

Див. визначення в файлі object.cpp, рядок 44

Граф викликів для цієї функції:



7.12.3.6 get_model() shared_ptr< Model > Object::get_model ( )

Повертає модель об'єкту.

Повертає

модель об'єкту

Див. визначення в файлі object.cpp, рядок 80

7.12.3.7 get_position() void Object::get_position (vec4 position) [override], [virtual]

Забирає поточну позицію об'єкта у просторі.

Аргументи

out	position	поточна позицію об'єкта

Переозначення з Bijective.

Див. визначення в файлі object.cpp, рядок 14

7.12.3.8 get_rotation() void Object::get_rotation (vec3 rotation) [override], [virtual]

Забирає поточний нахил об'єкта.

Аргументи

out	rotation	поточний нахил об'єкта

Переозначення з Bijective.

Див. визначення в файлі object.cpp, рядок 24

7.12.3.9 get_transform() mat4 * Object::get_transform ( )

Повертає матрицю лінійних перетворень без переміщення.

Повертає

матриця лінійних перетворень

Див. визначення в файлі object.cpp, рядок 115

7.12.3.10 rotate() void Object::rotate (enum coord_enum coord, float delta_time) [override], [virtual]

Обертає об'єкт.

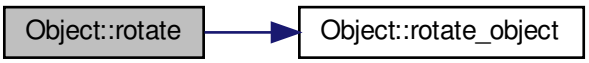
Аргументи

in	coord	позначає координатну вісь навколо якої обертати
in	delta_time	скільки часу пройшло з останнього кадру

Переозначення з Bijective.

Див. визначення в файлі object.cpp, рядок 65

Граф всіх викликів цієї функції:



7.12.3.11 rotate_all() void Object::rotate_all (vec3 angles) [override], [virtual]

Обернути об'єкт по всім осям.

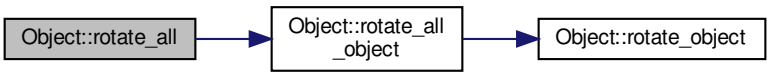
Аргументи

in	angles	вектор кутів у радіанах на кожну вісь

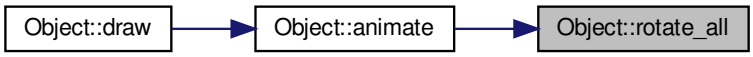
Переозначення з Bijective.

Див. визначення в файлі object.cpp, рядок 69

Граф всіх викликів цієї функції:



Граф викликів для цієї функції:



7.12.3.12 rotate_all_object() void Object::rotate_all_object (vec3 angles)

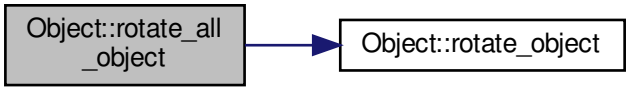
Обертає об'єкт по всім осям незалежно від часу.

Аргументи

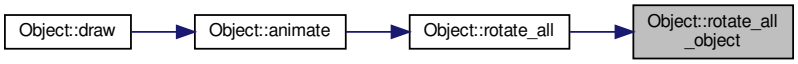
angles	вектор кутів обертання по трьом осям (x, y, z)

Див. визначення в файлі object.cpp, рядок 99

Граф всіх викликів цієї функції:



Граф викликів для цієї функції:



7.12.3.13 rotate_object()

void Object::rotate_object (
float angle,
enum coord_enum coord)

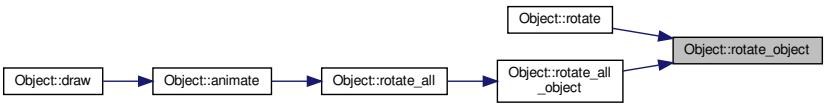
Обертає об'єкт по осі незалежно від часу.

Аргументи

angle	кут обертання у радіанах
coord	ввісь обертання

Див. визначення в файлі object.cpp, рядок 89

Граф викликів для цієї функції:



7.12.3.14 scale()

void Object::scale (
vec3 scale)

Збільщує або зменшує об'єкт по трьом осям.

Аргументи

scale	коеффіціенти зміни розміру по осям (x, y, z)

Див. визначення в файлі object.cpp, рядок 111

7.12.3.15 set_animation() void Object::set_animation (vec3 rotation) [override], [virtual]

Задає нову анімацію обернення об'єкта.

Аргументи

in	rotation	нова анімація обернення об'єкта.

Переозначення з Bijective.

Див. визначення в файлі object.cpp, рядок 39

7.12.3.16 set_position() void Object::set_position (vec4 position) [override], [virtual]

Задає нову позицію об'єкта у просторі.

Аргументи

in	position	нова позиція об'єкта у просторі

Переозначення з Bijective.

Див. визначення в файлі object.cpp, рядок 19

7.12.3.17 set_rotation() void Object::set_rotation (vec3 rotation) [override], [virtual]

Задає новий нахил об'єкта.

Аргументи

in	rotation	новий нахил об'єкта

Переозначення з Bijective.

Див. визначення в файлі object.cpp, рядок 29

7.12.3.18 translate() void Object::translate (vec3 distances, float delta_time) [override], [virtual]

Переміщує об'єкт у просторі.

Переміщує об'єкт у просторі на відстані з аргументу distances, де кожне значення вектору позначає відстань відповідної осі.

Аргументи

in	distances	відстані переміщення по осям
in	delta_time	скільки часу пройшло з останнього кадру

Переозначення з Bijective.

Див. визначення в файлі object.cpp, рядок 49

7.12.3.19 translate_object() void Object::translate_object (vec3 distances)

Переміщує об'єкт незалежно від часу.

Аргументи

distances	відстані переміщення по трьом осям (x, y, z)

Див. визначення в файлі object.cpp, рядок 85

7.12.4 Компонентні дані

7.12.4.1 animation vec4 Object::animation [private]

Анімація обертання по осям $(x, y, z, 1.0)$.

Див. визначення в файлі object.h, рядок 32

7.12.4.2 model shared_ptr<Model> Object::model [private]

Модель об'єкту.

Див. визначення в файлі object.h, рядок 36

7.12.4.3 position vec4 Object::position [private]

Позиція об'єкту у просторі

Див. визначення в файлі object.h, рядок 28

7.12.4.4 transform mat4 Object::transform [private]

Матриця лінійних перетворень.

Див. визначення в файлі object.h, рядок 24

Документація цих класів була створена з файлів:

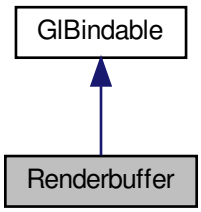
- osdo/object.h
- osdo/object.cpp

7.13 Клас Renderbuffer

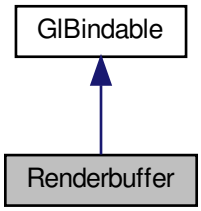
Буфер рендеренгу (для зберігання кольорів або глибини)

#include <renderbuffer.h>

Схема успадкувань для Renderbuffer



Діаграма зв'язків класу Renderbuffer:



Загальнодоступні елементи

- Renderbuffer ()
  Конструктор, що створює буфер рендеренгу.
- ~Renderbuffer () override
- void make_multisample (GLsizei size[2], GLenum target) const
  Створює буфер рендеренгу з згладжуванням.
- void make (GLsizei size[2], GLenum target) const
  Створює буфер рендеренгу.

Приватні елементи

- GLuint _generate () const override
  Віртуальний метод що створює буфер рендеренгу.
- virtual void _bind (const GLuint id, GLenum target) const override
  Віртуальний метод, що прив'язує контекст OpenGL до буферу рендеренгу.
- virtual GLenum _default () const override
  Віртуальний метод, що задає ціль прив'язки за замовчуванням.

Додаткові успадковані елементи

7.13.1 Детальний опис

Буфер рендеренгу (для зберігання кольорів або глибини)

Див. визначення в файлі renderbuffer.h, рядок 14

7.13.2 Конструктор(и)

7.13.2.1 Renderbuffer() Renderbuffer::Renderbuffer ( ) [inline]

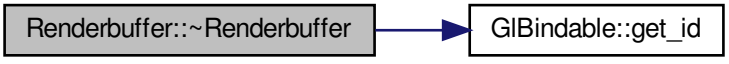
Конструктор, що створює буфер рендеренгу.

Див. визначення в файлі renderbuffer.h, рядок 36

7.13.2.2 ~Renderbuffer() Renderbuffer::~Renderbuffer ( ) [override]

Див. визначення в файлі renderbuffer.cpp, рядок 21

Граф всіх викликів цієї функції:



7.13.3 Опис методів компонент

7.13.3.1 _bind() void Renderbuffer::_bind (const GLuint id, GLenum target) const [override], [private], [virtual]

Віртуальний метод, що прив'язує контекст OpenGL до буферу рендеренгу.

Аргументи

id	індекс буфер рендеренгу
target	ціль прив'язки буфера рендеренгу

Переозначення з GlBindable.

Див. визначення в файлі renderbuffer.cpp, рядок 11

7.13.3.2 _default() GLenum Renderbuffer::_default ( ) const [override], [private], [virtual]

Віртуальний метод, що задає ціль прив'язки за замовчуванням.

Повертає

ціль прив'язки за замовчуванням

Переозначення з GlBindable.

Див. визначення в файлі renderbuffer.cpp, рядок 16

7.13.3.3 _generate() GLuint Renderbuffer::_generate ( ) const [override], [private], [virtual]

Віртуальний метод що створює буфер рендеренгу.

Повертає

індекс буфер рендеренгу

Переозначення з GlBindable.

Див. визначення в файлі renderbuffer.cpp, рядок 4

7.13.3.4 make() void Renderbuffer::make (GLsizei size[2], GLenum target) const

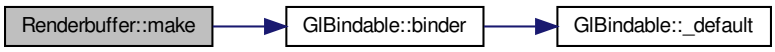
Створює буфер рендеренгу.

Аргументи

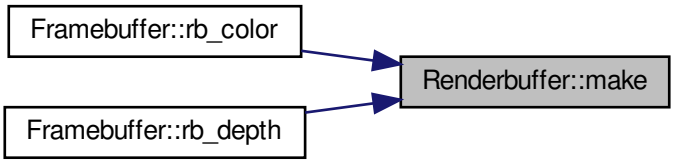
size	ширина та висота кадру
target	ціль буферу рендеренгу

Див. визначення в файлі renderbuffer.cpp, рядок 31

Граф всіх викликів цієї функції:



Граф викликів для цієї функції:



7.13.3.5 make_multisample() void Renderbuffer::make_multisample (
GLsizei size[2],
GLenum target) const

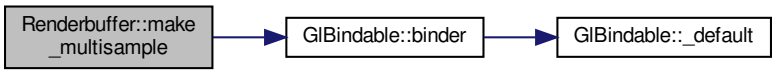
Створює буфер рендеренгу з згладжуванням.

Аргументи

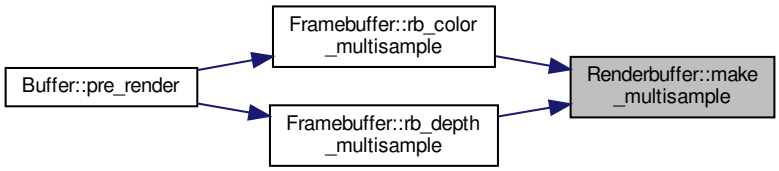
size	ширина та висота кадру
target	ціль буферу рендеренгу

Див. визначення в файлі renderbuffer.cpp, рядок 25

Граф всіх викликів цієї функції:



Граф викликів для цієї функції:



Документація цих класів була створена з файлів:

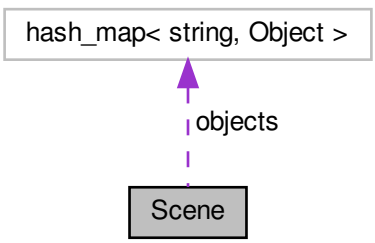
- osdo/renderbuffer.h
- osdo/renderbuffer.cpp

7.14 Структура Scene

Сцена із об'єктами.

#include <scene.h>

Діаграма зв'язків класу Scene:



Загальнодоступні елементи

- Scene (const Context::Models &objects)
  Конструктор, що створює об'єкти у сцені за заготовленими у агрументі objects

Загальнодоступні статичні елементи

- static shared_ptr< Scene > create (const Context::Models &objects)
  Створює сцену

Загальнодоступні атрибути

- hash_map< string, Object > objects
  Об'єкти у сцені.

7.14.1 Детальний опис

Сцена із об'єктами.

Див. визначення в файлі scene.h, рядок 16

7.14.2 Конструктор(и)

7.14.2.1 Scene() Scene::Scene (const Context::Models & objects)

Конструктор, що створює об'єкти у сцені за заготовленими у агрументі objects

Аргументи

objects	заготовлені об'єкти для додавання у сцену

Див. визначення в файлі scene.cpp, рядок 7

7.14.3 Опис методів компонент

7.14.3.1 create() shared_ptr< Scene > Scene::create (const Context::Models & objects) [static]

Створює сцену

Аргументи

objects	заготовлені об'єкти для додавання у сцену

Повертає

Розумний вказівник на об'єкт сцени.

Див. визначення в файлі scene.cpp, рядок 10

7.14.4 Компонентні дані

7.14.4.1 objects hash_map<string, Object> Scene::objects

Об'єкти у сцені.

Див. визначення в файлі scene.h, рядок 20

Документація цих структур була створена з файлів:

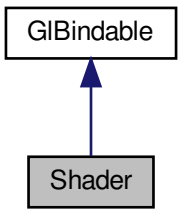
- osdo/scene.h
- osdo/scene.cpp

7.15 Клас Shader

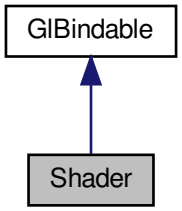
Клас взаємодії з шейдером у видеокарті.

#include <shader.h>

Схема успадкувань для Shader



Діаграма зв'язків класу Shader:



Загальнодоступні типи

- typedef hash_map< ShaderType, string > shader_map
  тип для зберігання шляхів файлів шейдеру по їх типам.

Загальнодоступні елементи

- Shader (const GLuint shader)
- ~Shader () override
- void set_bool (const char *name, bool value)
  Задати поле шейдеру типу bool
- void set_int (const char *name, int value)
  Задати поле шейдеру типу int
- void set_float (const char *name, float value)
  Задати поле шейдеру типу float
- void set_vec2 (const char *name, vec2 value)
  Задати поле шейдеру типу vec2
- void set_vec2f (const char *name, float x, float y)
  Задати вектор-поле шейдеру типу vec2
- void set_vec3 (const char *name, vec3 value)
  Задати поле шейдеру типу vec3
- void set_vec3f (const char *name, float x, float y, float z)
  Задати вектор-поле шейдеру типу vec3
- void set_vec4 (const char *name, vec4 value)
  Задати поле шейдеру типу vec4
- void set_vec4f (const char *name, float x, float y, float z, float w)
  Задати вектор-поле шейдеру типу vec4
- void set_mat2 (const char *name, mat2 mat)
  Задати поле шейдеру типу mat2
- void set_mat3 (const char *name, mat3 mat)
  Задати поле шейдеру типу mat3
- void set_mat4 (const char *name, mat4 mat)
  Задати поле шейдеру типу mat4

Загальнодоступні статичні елементи

- static shared_ptr< Shader > create (const shader_map &shaders_paths)
  Створює об'єкт шейдеру за заданими шляхами файлів шейдерів.

Приватні елементи

- virtual void _bind (const GLuint id, GLenum target) const override
  Віртуальний метод, що прив'язує контекст OpenGL до шейдеру.

Додаткові успадковані елементи

7.15.1 Детальний опис

Клас взаємодії з шейдером у видеокарті.

Див. визначення в файлі shader.h, рядок 31

7.15.2 Опис типів користувача

7.15.2.1 shader_map typedef hash_map<ShaderType, string> Shader::shader_map

тип для зберігання шляхів файлів шейдеру по їх типам.

Див. визначення в файлі shader.h, рядок 42

7.15.3 Конструктор(и)

7.15.3.1 Shader() Shader::Shader (const GLuint shader)

Див. визначення в файлі shader.cpp, рядок 118

7.15.3.2 ~Shader() Shader::~Shader ( ) [override]

Див. визначення в файлі shader.cpp, рядок 120

Граф всіх викликів цієї функції:

7.15.4 Опис методів компонент

7.15.4.1 _bind() void Shader::_bind (const GLuint id, GLenum target) const [override], [private], [virtual]

Віртуальний метод, що прив'язує контекст OpenGL до шейдеру.

Аргументи

id	індекс шейдеру OpenGL
target	не використовується, реалізован для інтерфейсу GlBindable::_bind

Переозначення з GlBindable.

Див. визначення в файлі shader.cpp, рядок 113

7.15.4.2 create() shared_ptr< Shader > Shader::create (const shader_map & shaders_paths) [static]

Створює об'єкт шейдеру за заданими шляхами файлів шейдерів.

Аргументи

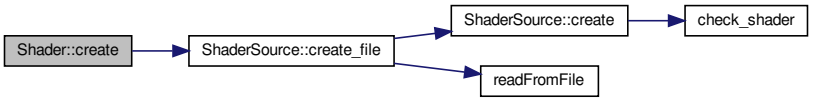
shaders_paths	шляхи файлів шейдеру по їх типам

Повертає

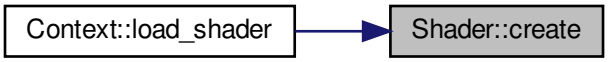
розумний вказівник на об'єкт шейдеру

Див. визначення в файлі shader.cpp, рядок 124

Граф всіх викликів цієї функції:



Граф викликів для цієї функції:



7.15.4.3 set_bool() void Shader::set_bool (const char * name, bool value)

Задати поле шейдеру типу bool

Аргументи

name	ім'я поля
value	значення поля

Див. визначення в файлі shader.cpp, рядок 134

Граф всіх викликів цієї функції:

7.15.4.4 set_float() void Shader::set_float (const char * name, float value)

Задати поле шейдеру типу float

Аргументи

name	ім'я поля
value	значення поля

Див. визначення в файлі shader.cpp, рядок 142

Граф всіх викликів цієї функції:

7.15.4.5 set_int()

void Shader::set_int (
const char * name,
int value)

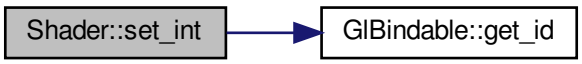
Задати поле шейдеру типу int

Аргументи

name	ім'я поля
value	значення поля

Див. визначення в файлі shader.cpp, рядок 138

Граф всіх викликів цієї функції:



7.15.4.6 set_mat2()

void Shader::set_mat2 (
const char * name,
mat2 mat)

Задати поле шейдеру типу mat2

Аргументи

name	ім'я поля
mat	значення поля

Див. визначення в файлі shader.cpp, рядок 176

Граф всіх викликів цієї функції:

7.15.4.7 set_mat3()

void Shader::set_mat3 (
const char * name,
mat3 mat)

Задати поле шейдеру типу mat3

Аргументи

name	ім'я поля
mat	значення поля

Див. визначення в файлі shader.cpp, рядок 181

Граф всіх викликів цієї функції:

7.15.4.8 set_mat4()

void Shader::set_mat4 (
const char * name,
mat4 mat)

Задати поле шейдеру типу mat4

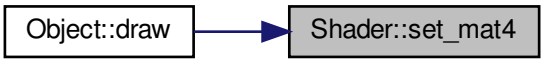
Аргументи

name	ім'я поля
mat	значення поля

Див. визначення в файлі shader.cpp, рядок 186

Граф всіх викликів цієї функції:

Граф викликів для цієї функції:



7.15.4.9 set_vec2() void Shader::set_vec2 (const char * name, vec2 value)

Задати поле шейдеру типу vec2

Аргументи

name	ім'я поля
value	значення поля

Див. визначення в файлі shader.cpp, рядок 146

Граф всіх викликів цієї функції:

7.15.4.10 set_vec2f() void Shader::set_vec2f (
const char * name,
float x,
float y)

Задати вектор-поле шейдеру типу vec2

Аргументи

name	ім'я поля
x	значення першого элементу вектор-поля
y	значення другого элементу вектор-поля

Див. визначення в файлі shader.cpp, рядок 151

Граф всіх викликів цієї функції:

7.15.4.11 set_vec3() void Shader::set_vec3 (
const char * name,
vec3 value)

Задати поле шейдеру типу vec3

Аргументи

name	ім'я поля
value	значення поля

Див. визначення в файлі shader.cpp, рядок 156

Граф всіх викликів цієї функції:

7.15.4.12 set_vec3f()

```
void Shader::set_vec3f (
    const char * name,
    float x,
    float y,
    float z )
```

Задати вектор-поле шейдеру типу vec3

Аргументи

name	ім'я поля
x	значення першого элементу вектор-поля
y	значення другого элементу вектор-поля
z	значення третього элементу вектор-поля

Див. визначення в файлі shader.cpp, рядок 161

Граф всіх викликів цієї функції:

7.15.4.13 set_vec4()

```
void Shader::set_vec4 (
    const char * name,
    vec4 value )
```

Задати поле шейдеру типу vec4

Аргументи

name	ім'я поля
value	значення поля

Див. визначення в файлі shader.cpp, рядок 166

Граф всіх викликів цієї функції:

7.15.4.14 set_vec4f()

```
void Shader::set_vec4f (
    const char * name,
    float x,
    float y,
    float z,
    float w )
```

Задати вектор-поле шейдеру типу vec4

Аргументи

name	ім'я поля
x	значення першого елементу вектор-поля
y	значення другого елементу вектор-поля
z	значення третього елементу вектор-поля
w	значення четвертого елементу вектор-поля

Див. визначення в файлі shader.cpp, рядок 171

Граф всіх викликів цієї функції:

Документація цих класів була створена з файлів:

- osdo/shader.h
- osdo/shader.cpp

7.16 Клас ShaderSource

Загальнодоступні елементи

- ShaderSource (const GLuint id)
- GLuint get_id ()
- void attach (const GLuint program)

Загальнодоступні статичні елементи

- static shared_ptr< ShaderSource > create (GLenum type, const char *code)
- static shared_ptr< ShaderSource > create_file (GLenum type, const string &path)

Приватні дані

- const GLuint id

7.16.1 Детальний опис

Див. визначення в файлі shader.cpp, рядок 70

7.16.2 Конструктор(и)

7.16.2.1 ShaderSource() ShaderSource::ShaderSource (const GLuint id) [inline]

Див. визначення в файлі shader.cpp, рядок 73

7.16.3 Опис методів компонент

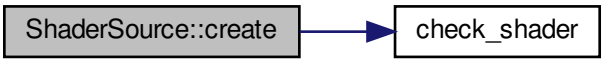
7.16.3.1 attach() void ShaderSource::attach (const GLuint program) [inline]

Див. визначення в файлі shader.cpp, рядок 90

7.16.3.2 create() static shared_ptr<ShaderSource> ShaderSource::create (
GLenum type,
const char * code) [inline], [static]

Див. визначення в файлі shader.cpp, рядок 74

Граф всіх викликів цієї функції:

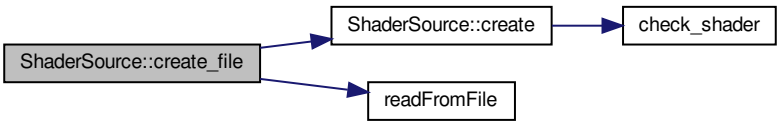


Граф викликів для цієї функції:

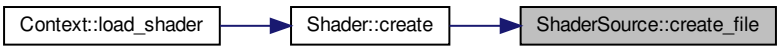
7.16.3.3 create_file() static shared_ptr<ShaderSource> ShaderSource::create_file (
GLenum type,
const string & path) [inline], [static]

Див. визначення в файлі shader.cpp, рядок 83

Граф всіх викликів цієї функції:



Граф викликів для цієї функції:



7.16.3.4 get_id() GLuint ShaderSource::get_id ( ) [inline]

Див. визначення в файлі shader.cpp, рядок 89

7.16.4 Компонентні дані

7.16.4.1 id const GLuint ShaderSource::id [private]

Див. визначення в файлі shader.cpp, рядок 71

Документація цього класу була створена з файлу:

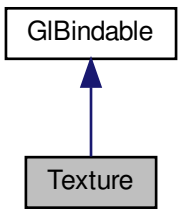
- osdo/shader.cpp

7.17 Клас Texture

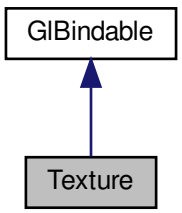
Клас текстури, що зберігаэться у відеокарті.

#include <texture.h>

Схема успадкувань для Texture



Діаграма зв'язків класу Texture:



Загальнодоступні елементи

- Texture ()
- ~Texture () override
- void update (const Image &image) const

  Завантажує зображення у текстуру.
- void make_2d_multisample (GLsizei size[2]) const

  Створює текстуру з згладжуванням.
- void make_2d (GLsizei size[2]) const

  Створює текстуру.

Приватні елементи

- GLuint _generate () const override

  Віртуальний метод що створює текстуру.
- virtual void _bind (const GLuint id, GLenum target) const override

  Віртуальний метод, що прив'язує контекст OpenGL до текстури.
- virtual GLenum _default () const override

  Віртуальний метод, що задає ціль прив'язки за замовчуванням.

Додаткові успадковані елементи

7.17.1 Детальний опис

Клас текстури, що зберігаэться у відеокарті.

Див. визначення в файлі texture.h, рядок 16

7.17.2 Конструктор(и)

7.17.2.1 Texture() Texture::Texture ( ) [inline]

Див. визначення в файлі texture.h, рядок 35

7.17.2.2 ~Texture() Texture::~Texture ( ) [override]

Див. визначення в файлі texture.cpp, рядок 22

Граф всіх викликів цієї функції:

7.17.3 Опис методів компонент

7.17.3.1 _bind() void Texture::_bind (const GLuint id, GLenum target) const [override], [private], [virtual]

Віртуальний метод, що прив'язує контекст OpenGL до текстури.

Аргументи

id	індекс текстури
target	ціль прив'язки текстури

Переозначення з GlBindable.

Див. визначення в файлі texture.cpp, рядок 12

7.17.3.2 _default() GLenum Texture::_default ( ) const [override], [private], [virtual]

Віртуальний метод, що задає ціль прив'язки за замовчуванням.

Повертає

ціль прив'язки за замовчуванням

Переозначення з GlBindable.

Див. визначення в файлі texture.cpp, рядок 17

7.17.3.3 _generate() GLuint Texture::_generate ( ) const [override], [private], [virtual]

Віртуальний метод що створює текстуру.

Повертає

індекс текстури

Переозначення з GlBindable.

Див. визначення в файлі texture.cpp, рядок 5

7.17.3.4 make_2d() void Texture::make_2d (GLsizei size[2]) const

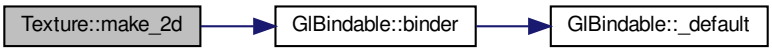
Створює текстуру.

Аргументи

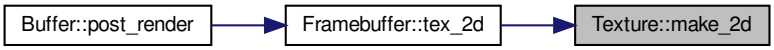
size	ширина та висота кадру

Див. визначення в файлі texture.cpp, рядок 53

Граф всіх викликів цієї функції:



Граф викликів для цієї функції:



7.17.3.5 make_2d_multisample() void Texture::make_2d_multisample (GLsizei size[2]) const

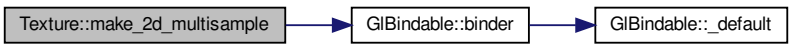
Створює текстуру з згладжуванням.

Аргументи

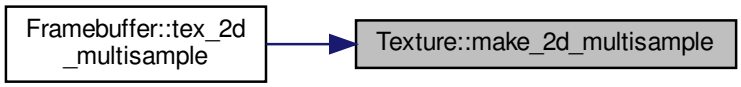
size	ширина та висота кадру

Див. визначення в файлі texture.cpp, рядок 47

Граф всіх викликів цієї функції:



Граф викликів для цієї функції:



7.17.3.6 update() void Texture::update (const Image & image) const

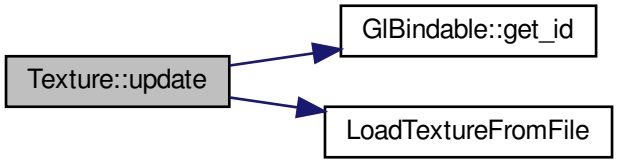
Завантажує зображення у текстуру.

Аргументи

image	зображення

Див. визначення в файлі texture.cpp, рядок 43

Граф всіх викликів цієї функції:



Документація цих класів була створена з файлів:

- osdo/texture.h
- osdo/texture.cpp

7.18 Шаблон класу OSDO::vector< T >

Вектор що не змінює свій розмір.

#include <easyvector.h>

Загальнодоступні елементи

- vector (size_t size=0)
  Конструктор з параметром розміру масиву
- vector (vector &&vector)
  Конструктор переносу
- vector (const vector &vector)
  Конструктор копіювання
- ~vector ()
- vector & operator= (vector &&vector)
- vector & operator= (const vector &vector)
- T & operator[] (size_t i)
- size_t size () const
- T * data ()
- const T * data () const
- void clear ()

Приватні елементи

- void _allocate (size_t size)
  Виділяє масив розміру size
- void _free ()
  Звільнює пам'ять від масиву.
- void _copy (const vector &vector)
  Копіює масив.
- void _move (vector &vector)
  Переміщює данні з іншого масиву.

Приватні дані

- T * arr
  Масив із елементами типу T
- size_t _size
  Поточний розмір масиву.

7.18.1 Детальний опис

template<class T>
class OSDO::vector< T >

Вектор що не змінює свій розмір.

Див. визначення в файлі easyvector.h, рядок 19

7.18.2 Конструктор(и)

7.18.2.1 vector() [1/3] template<class T >

OSDO::vector< T >::vector (
size_t size = 0) [inline]

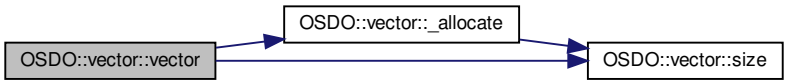
Конструктор з параметром розміру масиву

Аргументи

size	розмір массиву

Див. визначення в файлі easyvector.h, рядок 71

Граф всіх викликів цієї функції:



Граф викликів для цієї функції:

7.18.2.2 vector() [2/3] template<class T > OSDO::vector< T >::vector (vector< T > && vector) [inline]

Конструктор переносу

Аргументи

vector	інший масив

Див. визначення в файлі easyvector.h, рядок 78

Граф всіх викликів цієї функції:

7.18.2.3 vector() [3/3] template<class T >

OSDO::vector< T >::vector (
const vector< T > & vector) [inline]

Конструктор копіювання

Аргументи

vector	інший масив

Див. визначення в файлі easyvector.h, рядок 85

Граф всіх викликів цієї функції:

7.18.2.4 ~vector() template<class T >

OSDO::vector< T >::~vector ( ) [inline]

Див. визначення в файлі easyvector.h, рядок 88

Граф всіх викликів цієї функції:

7.18.3 Опис методів компонент

7.18.3.1 _allocate() template<class T >

void OSDO::vector< T >::_allocate (
size_t size) [inline], [private]

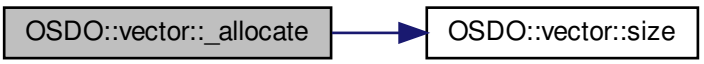
Виділяє масив розміру size

Аргументи

size	розмір масиву

Див. визначення в файлі easyvector.h, рядок 32

Граф всіх викликів цієї функції:



Граф викликів для цієї функції:

7.18.3.2 _copy() template<class T >

void OSDO::vector< T >::_copy (
const vector< T > & vector) [inline], [private]

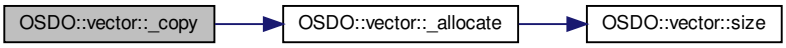
Копіює масив.

Аргументи

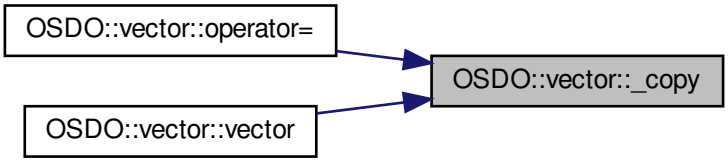
vector	інший масив

Див. визначення в файлі easyvector.h, рядок 53

Граф всіх викликів цієї функції:



Граф викликів для цієї функції:



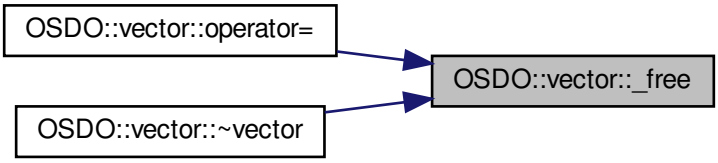
7.18.3.3 _free() template<class T >

void OSDO::vector< T >::_free ( ) [inline], [private]

Звільнює пам'ять від масиву.

Див. визначення в файлі easyvector.h, рядок 43

Граф викликів для цієї функції:



7.18.3.4 _move() template<class T >

void OSDO::vector< T >::_move (
vector< T > & vector) [inline], [private]

Переміщює данні з іншого масиву.

Аргументи

vector	інший масив

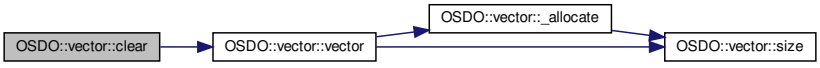
Див. визначення в файлі easyvector.h, рядок 62

7.18.3.5 clear() template<class T >

void OSDO::vector< T >::clear ( ) [inline]

Див. визначення в файлі easyvector.h, рядок 111

Граф всіх викликів цієї функції:



7.18.3.6 data() [1/2] template<class T >

T* OSDO::vector< T >::data ( ) [inline]

Див. визначення в файлі easyvector.h, рядок 105

7.18.3.7 data() [2/2] template<class T >

const T* OSDO::vector< T >::data ( ) const [inline]

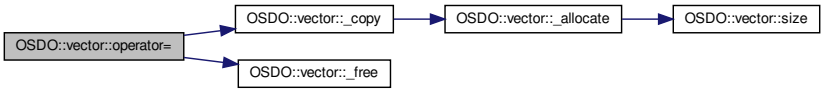
Див. визначення в файлі easyvector.h, рядок 108

7.18.3.8 operator=() [1/2] template<class T >

vector& OSDO::vector< T >::operator= (
const vector< T > & vector) [inline]

Див. визначення в файлі easyvector.h, рядок 94

Граф всіх викликів цієї функції:

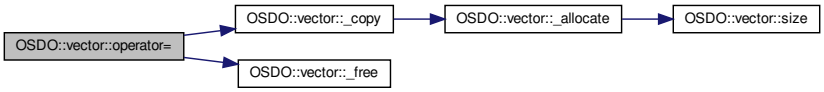


7.18.3.9 operator=() [2/2] template<class T >

vector& OSDO::vector< T >::operator= (
vector< T > && vector) [inline]

Див. визначення в файлі easyvector.h, рядок 89

Граф всіх викликів цієї функції:



7.18.3.10 operator[]() template<class T >

T& OSDO::vector< T >::operator[] (
size_t i) [inline]

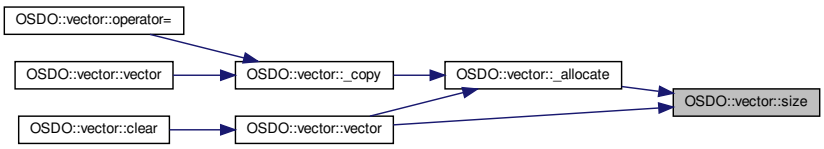
Див. визначення в файлі easyvector.h, рядок 99

7.18.3.11 size() template<class T >

size_t OSDO::vector< T >::size ( ) const [inline]

Див. визначення в файлі easyvector.h, рядок 102

Граф викликів для цієї функції:



7.18.4 Компонентні дані

7.18.4.1 _size template<class T >

size_t OSDO::vector< T >::_size [private]

Поточний розмір масиву.

Див. визначення в файлі easyvector.h, рядок 27

7.18.4.2 arr template<class T >

T* OSDO::vector< T >::arr [private]

Масив із елементами типу T

Див. визначення в файлі easyvector.h, рядок 23

Документація цього класу була створена з файлу:

- osdo/easyvector.h

7.19 Структура Vertex

Структура вершини, для передачі у відеокарту.

#include <vertex.h>

Загальнодоступні атрибути

- vec4 position
  Позиція вершини у просторі.
- vec3 normal
  Нормаль вершини.
- unsigned char color [4]
  Колір вершини.
- vec2 uv
  Координати вершини на текстурі.

7.19.1 Детальний опис

Структура вершини, для передачі у відеокарту.

Див. визначення в файлі vertex.h, рядок 12

7.19.2 Компонентні дані

7.19.2.1 color unsigned char Vertex::color[4]

Колір вершини.

Див. визначення в файлі vertex.h, рядок 24

7.19.2.2 normal vec3 Vertex::normal

Нормаль вершини.

Див. визначення в файлі vertex.h, рядок 20

7.19.2.3 position vec4 Vertex::position

Позиція вершини у просторі.

Див. визначення в файлі vertex.h, рядок 16

7.19.2.4 uv vec2 Vertex::uv

Координати вершини на текстурі.

Див. визначення в файлі vertex.h, рядок 28

Документація цієї структури була створена з файлу:

- osdo/vertex.h

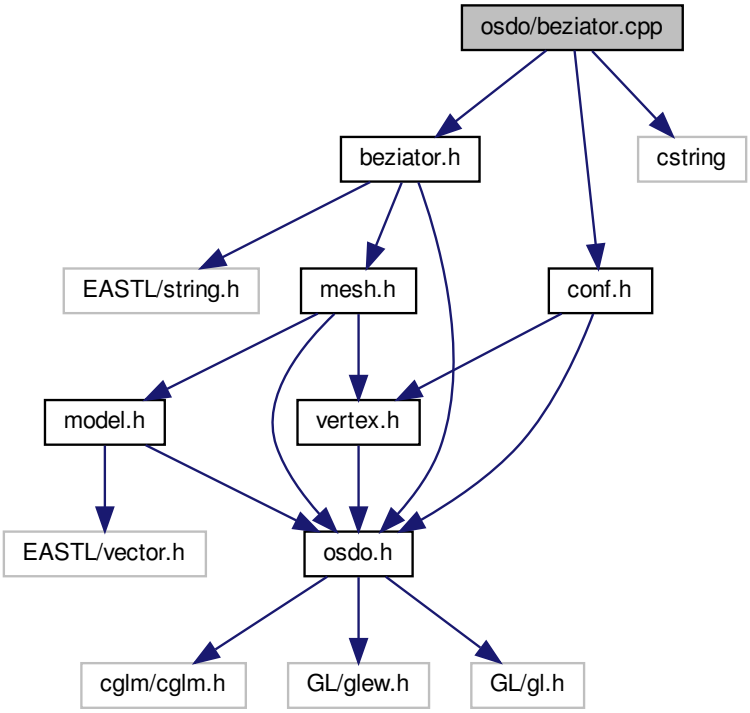
8 Файли

8.1 Файл LICENSE.md

8.2 Файл osdo/beziator.cpp

```
#include "beziator.h"
#include "conf.h"
```

#include <cstring>
Діаграма включених заголовочних файлів для beziator.cpp:



Макровизначення

- #define BEZIER_TANGENT_INIT
- #define ucast static_cast<unsigned>

Визначення типів

- typedef Vertex * surface_t[4][4]

Функції

- void bezier_curve (float a, mat4 points, vec4 dest)
- void bezier_curve_tangent (float a, mat4 points, vec4 dest)
- void bezier_surface (float u, float v, surface_t points, vec4 dest, vec4 normal)

8.2.1 Опис макровизначень

8.2.1.1 BEZIER_TANGENT_INIT #define BEZIER_TANGENT_INIT

Макровизначення:
```
{\
{  0,  0,  0,  0},\
{ -3,  9, -9,  3},\
{  6,-12,  6,  0},\
{ -3,  3,  0,  0},}
```

Див. визначення в файлі beziator.cpp, рядок 5

8.2.1.2 ucast #define ucast static_cast<unsigned>

Див. визначення в файлі beziator.cpp, рядок 11

8.2.2 Опис визначень типів

8.2.2.1 surface_t typedef Vertex* surface_t[4][4]

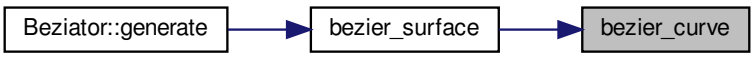
Див. визначення в файлі beziator.cpp, рядок 16

8.2.3 Опис функцій

8.2.3.1 bezier_curve() void bezier_curve (
float a,
mat4 points,
vec4 dest)

Див. визначення в файлі beziator.cpp, рядок 71

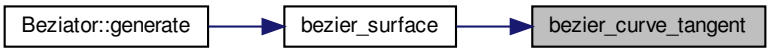
Граф викликів для цієї функції:



8.2.3.2 bezier_curve_tangent() void bezier_curve_tangent (
float a,
mat4 points,
vec4 dest)

Див. визначення в файлі beziator.cpp, рядок 78

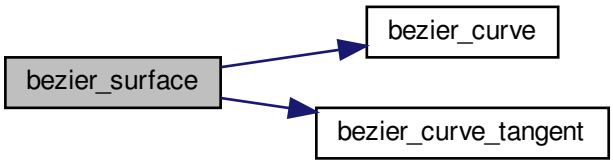
Граф викликів для цієї функції:



8.2.3.3 bezier_surface() void bezier_surface (
float u,
float v,
surface_t points,
vec4 dest,
vec4 normal)

Див. визначення в файлі beziator.cpp, рядок 85

Граф всіх викликів цієї функції:



Граф викликів для цієї функції:

8.3 beziator.cpp

```
00001 #include "beziator.h"
00002 #include "conf.h"
00003 #include <cstring>
00004
00005 #define BEZIER_TANGENT_INIT {\
00006 {  0,  0,  0,  0},\
00007 { -3,  9, -9,  3},\
00008 {  6,-12,  6,  0},\
00009 { -3,  3,  0,  0},}
00010
00011 #define ucast static_cast<unsigned>
00012
00013 static mat4 BEZIER = GLM_BEZIER_MAT_INIT;
00014 static mat4 BEZIER_TANGENT = BEZIER_TANGENT_INIT;
00015
00016 typedef Vertex *surface_t[4][4];
00017
00018 Beziator::Beziator(const string& path) : path(path) {}
00019
00020 bool Beziator::init() {
00021     printf("%s\n", path.c_str());
00022     FILE *file = fopen(path.c_str(), "r");
00023     if (file == nullptr) {
00024         return false;
00025     }
00026
00027     size_t points_size, surfaces_size;
00028     fscanf(file, "%lu%lu", &points_size, &surfaces_size);
00029     vector<Vertex> &points = vertices;
00030     points.resize(points_size);
00031     indices.resize(surfaces_size * 16);
00032     surfacei_t *surfaces = reinterpret_cast<surfacei_t*>(indices.data());
00033
00034     points_size = points.size();
00035     unsigned char color[4] = {0, 255, 0, 255};
00036     surfaces_size = indices.size() / 16;
00037     for (size_t i = 0; i < points_size; i++) {
00038         vec4 init = GLM_VEC4_BLACK_INIT;
00039         vec4 &point = points[i].position;
00040         glm_vec4_copy(init, point);
00041         fscanf(file, "%f%f%f", point, point + 1, point + 2);
00042         memcpy(points[i].color, color, 4);
00043         memcpy(points[i].normal, point, 3 * sizeof(float));
00044     }
00045     int j, k;
00046     for (size_t i = 0; i < surfaces_size; i++) {
00047         for (j = 0; j < 4; j++)
00048             for (k = 0; k < 4; k++) {
00049                 fscanf(file, "%u", surfaces[i][j] + k);
00050             }
00051     }
00052     fclose(file);
00053
00054     update(points.data(), points.size(), indices.data(), indices.size());
00055     return true;
00056 }
00057
00058 Beziator::~Beziator() {
00059 }
00060
00061 void Beziator::draw(Shader &shader, bool pre_generated) {
00062     if (pre_generated) {
00063         this->mesh.draw_mode(GL_TRIANGLES);
00064     } else {
00065         glPatchParameteri(GL_PATCH_VERTICES, 16);
00066         Mesh::draw_mode(GL_PATCHES);
00067     }
00068 }
00069
00070
00071 void bezier_curve(float a, mat4 points, vec4 dest) {
00072     mat4 matrix;
00073     glm_vec4_cubic(a, dest);
00074     glm_mat4_mul(points, BEZIER, matrix);
00075     glm_mat4_mulv(matrix, dest, dest);
00076 }
00077
00078 void bezier_curve_tangent(float a, mat4 points, vec4 dest) {
00079     mat4 matrix;
00080     glm_vec4_cubic(a, dest);
00081     glm_mat4_mul(points, BEZIER_TANGENT, matrix);
00082     glm_mat4_mulv(matrix, dest, dest);
00083 }
00084
00085 void bezier_surface(
```

```
00086         float u, float v, surface_t points, vec4 dest, vec4 normal) {
00087     mat4 m, res1, res2, res3;
00088
00089     for (int i = 0; i < 4; i++) {
00090         glm_vec4_copy(points[0][i]->position, m[0]);
00091         glm_vec4_copy(points[1][i]->position, m[1]);
00092         glm_vec4_copy(points[2][i]->position, m[2]);
00093         glm_vec4_copy(points[3][i]->position, m[3]);
00094         bezier_curve(u, m, res1[i]);
00095
00096         glm_vec4_copy(points[i][0]->position, m[0]);
00097         glm_vec4_copy(points[i][1]->position, m[1]);
00098         glm_vec4_copy(points[i][2]->position, m[2]);
00099         glm_vec4_copy(points[i][3]->position, m[3]);
00100         bezier_curve(v, m, res2[i]);
00101     }
00102
00103     bezier_curve(v, res1, dest);
00104     bezier_curve_tangent(v, res1, res3[1]);
00105     bezier_curve_tangent(u, res2, res3[3]);
00106
00107     glm_cross(res3[1], res3[3], normal);
00108 }
00109
00110 bool Beziator::save() {
00111     FILE *file = fopen(this->path.c_str(), "w");
00112     if (file == nullptr) {
00113         printf("ERROR: failed to open file %s\n", this->path.c_str());
00114         return false;
00115     }
00116     size_t surfaces_size = this->indices.size() / 16;
00117     surfacei_t *surfaces = reinterpret_cast<surfacei_t*>(indices.data());
00118     fprintf(file, "%lu %lu\n", this->vertices.size(), this->indices.size() / 16);
00119     for (size_t i = 0; i < this->vertices.size(); i++) {
00120         vec4 &point = this->vertices[i].position;
00121         fprintf(file, "%f %f %f\n", static_cast<double>(point[0]),
00122                 static_cast<double>(point[1]), static_cast<double>(point[2]));
00123     }
00124     int j, k;
00125     for (size_t i = 0; i < surfaces_size; i++) {
00126         for (j = 0; j < 4; j++)
00127             for (k = 0; k < 4; k++) {
00128                 fprintf(file, "%u ", surfaces[i][j][k]);
00129             }
00130         fprintf(file, "\n");
00131     }
00132     fclose(file);
00133     return true;
00134 }
00135
00136 void Beziator::generate(size_t d) {
00137     static const int controls_lines[][2] = {
00138         {0, 0}, {0, 1}, {0, 0}, {1, 1}, {0, 0}, {1, 0},
00139         {0, 3}, {0, 2}, {0, 3}, {1, 2}, {0, 3}, {1, 3},
00140         {3, 0}, {2, 0}, {3, 0}, {2, 1}, {3, 0}, {3, 1},
00141         {3, 3}, {3, 2}, {3, 3}, {2, 2}, {3, 3}, {2, 3},
00142     };
00143     static const int ctrls_size = sizeof(controls_lines) / sizeof(int[2]);
00144     /* // Old variant of config, I leave it for several commits
00145     static const uint8_t ALL_SQUARE_LINES[][4] = {
00146         {1, 0, 0, 0}, {0, 0, 0, 1}, {0, 1, 1, 1}, {1, 1, 1, 0},
00147         {1, 1, 0, 0}, {0, 0, 1, 1}, {1, 0, 0, 1}, {0, 1, 1, 0}
00148     };*/
00149     static const uint8_t SQUARE_TYPES[][10][2] = {
00150         {{0, 0}, {0, 1}, {1, 1}, {0, 0}, {8, 8},
00151          {1, 1}, {1, 0}, {0, 0}, {1, 1}, {9, 9}},
00152         {{0, 0}, {0, 1}, {1, 1}, {1, 0}, {0, 0}, {9, 9}},
00153         {{1, 0}, {0, 0}, {0, 1}, {1, 0}, {8, 8},
00154          {0, 1}, {1, 1}, {1, 0}, {0, 1}, {9, 9}},
00155         /* // And again old variant of config
00156         {1, 2, 4, 1, 8, 0, 5, 3, 0, 9},
00157         {0, 1, 2, 3, 0, 9},
00158         {0, 1, 5, 0, 8, 2, 3, 5, 2, 9},*/
00159     };
00160     static const uint8_t BEZIER_SQUARE_TYPES[3][3] = {
00161         {0, 1, 2}, {1, 1, 1}, {2, 1, 0}
00162     };
00163
00164     size_t j, k, index;
00165     float x, u, v;
00166     vec4 *point, vertex, normal;
00167     surface_t surface;
00168     GLuint verts = 0, verts2 = 0;
00169     //verts3 = 0;
00170     const int *c;
00171     mat4 m4b;
00172     uint8_t si, sj;
```

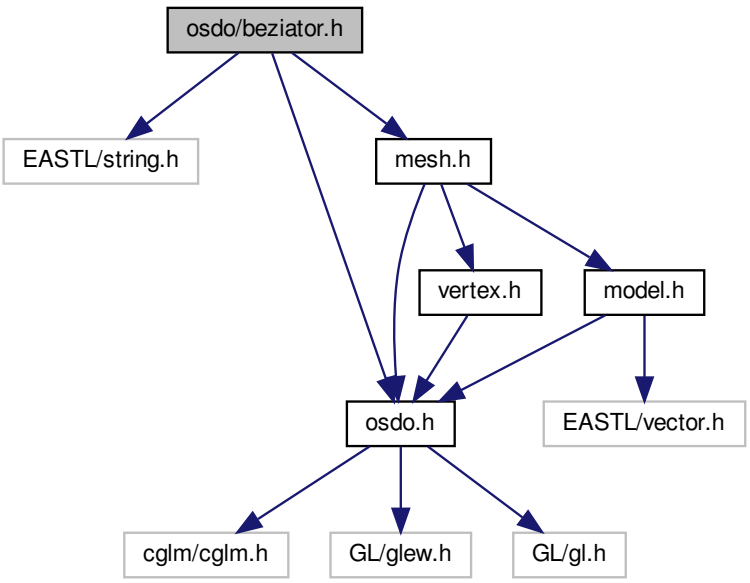
```
00173     const uint8_t (*st)[2];
00174
00175     Mesh *mesh = &this->mesh;
00176     // Mesh *mesh_skel = &this->frame;
00177     // Mesh *mesh_normals = &this->normals;
00178     x = 1.f / (d - 1);
00179
00180     const size_t surfaces_size = this->indices.size() / 16;
00181     surfacei_t *surfaces = reinterpret_cast<surfacei_t*>(indices.data());
00182     const size_t size = 6 * 9 * d * d * surfaces_size;
00183     //const GLsizei sizei = static_cast<GLsizei>(size);
00184     vector<Vertex> V(size);
00185     vector<GLuint> E(size);
00186     /*vector<Vertex> V2(size);
00187     vector<GLuint> E2(size);
00188     vector<Vertex> V3(this->vertices.size());
00189     vector<GLuint> E3(this->vertices.size() * 4);*/
00190
00191     // Creator frame vertices
00192     /*for (size_t i = 0; i < this->vertices.size(); i++) {
00193         point = &this->vertices[i].position;
00194         glm_vec3_copy(*point, V2[i].position);
00195         V2[i].color[1] = 255;
00196         V2[i].color[3] = 255;
00197     }*/
00198
00199     for (size_t i = 0; i < surfaces_size; i++) {
00200         for (j = 0; j < 4; j++) {
00201             for (k = 0; k < 4; k++) {
00202                 surface[j][k] = &(this->vertices[surfaces[i][j][k]]);
00203             }
00204         }
00205         // Creator frame lines
00206         for (j = 0; j < ctrls_size; j++) {
00207             c = controls_lines[j];
00208             //E2[verts2++] = ucast(surfaces[i][c[0]][c[1]]);
00209         }
00210
00211         // Create vertices
00212         for (j = 0; j < d; j++) {
00213             for (k = 0; k < d; k++) {
00214                 u = static_cast<float>(j)*x; v = static_cast<float>(k)*x;
00215                 index = i * d * d + j * d + k;
00216                 bezier_surface(u, v, surface, vertex, normal);
00217                 glm_normalize(normal);
00218                 glm_vec3_copy(vertex, V[index].position);
00219                 glm_vec3_copy(normal, V[index].normal);
00220                 V[index].color[0] = 0;
00221                 V[index].color[1] = 255;
00222                 V[index].color[2] = 0;
00223                 /*glm_vec3_copy(vertex, V3[verts3].position);
00224                 E3[verts3] = verts3;
00225                 verts3++;
00226                 glm_vec3_add(normal, vertex, V3[verts3].position);
00227                 E3[verts3] = verts3;
00228                 verts3++;*/
00229             }
00230         }
00231
00232         // Create triangles
00233         for (j = 0; j < d - 1; j++)
00234             for (k = 0; k < d - 1; k++) {
00235                 E[verts++] = ucast(i * d * d + (j + 1) * d + k);
00236                 E[verts++] = ucast(i * d * d + j * d + k);
00237                 E[verts++] = ucast(i * d * d + j * d + k + 1);
00238
00239                 E[verts++] = ucast(i * d * d + j * d + k + 1);
00240                 E[verts++] = ucast(i * d * d + (j + 1) * d + k + 1);
00241                 E[verts++] = ucast(i * d * d + (j + 1) * d + k);
00242             }
00243
00244         for (si = 0; si < 3; si++) {
00245             for (sj = 0; sj < 3; sj++) {
00246                 st = SQUARE_TYPES[BEZIER_SQUARE_TYPES[si][sj]];
00247                 while (st[2][0] != 9) {
00248                     if (st[2][0] == 8) {
00249                         st += 3;
00250                     }
00251                     index = static_cast<size_t>((surface)[si+st[1][0]][sj+st[1][1]] - this->vertices.data());
00252                     glm_vec3_sub(((surface)[si+st[1][0]][sj+st[1][1]])->position,
00253                             ((surface)[si+st[0][0]][sj+st[0][1]])->position, m4b[0]);
00254                     glm_vec3_sub(((surface)[si+st[1][0]][sj+st[1][1]])->position,
00255                             (surface[si+st[2][0]][sj+st[2][1]])->position, m4b[1]);
00256                     glm_vec3_cross(m4b[0], m4b[1], m4b[2]);
00257                     //glm_vec3_add(V2[index].normal, m4b[2], V2[index].normal);
00258                     st++;
00259                 }
```

```
00260             }
00261         }
00262     }
00263     /*
00264     // Example drawing of normals for frame
00265     for (size_t i = 0; i < this->points_size; i++) {
00266         glm_normalize(V2[i].normal);
00267         glm_vec3_add(V2[i].position, V2[i].normal,
00268                      V2[i+this->points_size].position);
00269         E2[verts2++] = (unsigned)i;
00270         E2[verts2++] = (unsigned)(i+this->points_size);
00271     }*/
00272     mesh->update(V.data(), V.size(), E.data(), E.size());
00273     //mesh_skel->update(V2, E2);
00274     //mesh_update(mesh_normals, sizei, sizei, V3, E3);
00275 }
00276
00277 void Beziator::rotate(size_t i) {
00278     surfacei_t s;
00279     surfacei_t *surfaces = reinterpret_cast<surfacei_t*>(indices.data());
00280     memcpy(s, surfaces[i], sizeof(surfacei_t));
00281     for (int k = 0; k < 4; k++)
00282         for (int j = 0; j < 4; j++) {
00283             surfaces[i][k][j] = s[j][k];
00284         }
00285 }
00286
00287 vector<Vertex> *Beziator::get_vertices() {
00288     return &vertices;
00289 }
```

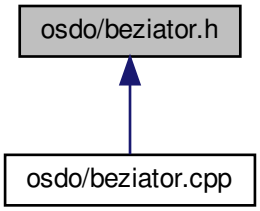
8.4 Файл osdo/beziator.h

Клас який зберігає та оброблює модель утворену через поверхні Безьє.

#include <EASTL/string.h>
#include "osdo.h"
#include "mesh.h"
Діаграма включених заголовочних файлів для beziator.h:



Граф файлів, які включають цей файл:



Класи

- class Beziator

  Клас який зберігає та оброблює модель утвореню через поверхні Безьє.

Визначення типів

- typedef GLuint surfacei_t[4][4]

  Набір індексів на вершини, що утворюють поверхню 4x4.

8.4.1 Детальний опис

Клас який зберігає та оброблює модель утвореню через поверхні Безьє.

Див. визначення в файлі beziator.h

8.4.2 Опис визначень типів

8.4.2.1 surfacei_t typedef GLuint surfacei_t[4][4]

Набір індексів на вершини, що утворюють поверхню 4x4.

Див. визначення в файлі beziator.h, рядок 17

8.5 beziator.h

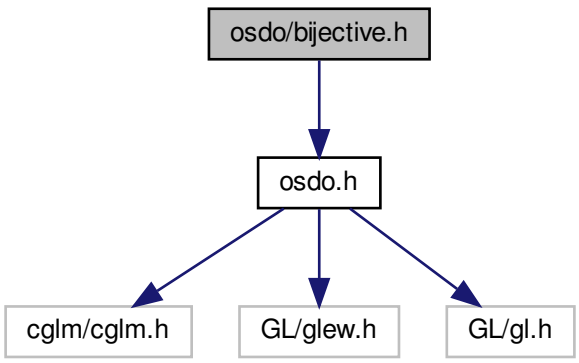
```
00001 /**
00002  * @file beziator.h
00003  * @brief Клас який зберігає та оброблює модель утворену через поверхні Безьє.
00004  */
00005 #ifndef BEZIATOR_H
00006 #define BEZIATOR_H
00007
00008 #include <EASTL/string.h>
00009 #include "osdo.h"
00010 #include "mesh.h"
00011
00012 using eastl::string;
00013
00014 /**
00015  * @brief Набір індексів на вершини, що утворюють поверхню 4x4.
00016  */
00017 typedef GLuint surfacei_t[4][4];
00018
00019 /**
00020  * @brief Клас який зберігає та оброблює модель утворену через поверхні Безьє.
00021  */
00022 class Beziator : public Mesh {
00023 public:
00024     /**
00025      * @brief Тип позначаючий вказівнкик на массив з поверхнями Безьє.
00026      */
00027     typedef surfacei_t* surfaces_vector;
00028 protected:
00029     /**
00030      * @brief Шлях до файлу у якому зберігається модель.
00031      */
00032     const string path;
00033     /**
00034      * @brief Згенерований за допомогою CPU меш моделі.
00035      */
00036     Mesh mesh;
00037     //Mesh frame;
00038     //Mesh normals;
00039     /**
00040      * @brief Масив вершин/вузлів моделі.
00041      */
00042     vector<Vertex> vertices;
00043     /**
00044      * @brief Масив індексів, що утворюють поверхні Безьє.
00045      *
00046      * Індекси розташовані у масиві по 16 элементів, які утворюють поверхню
00047      * з контрольними точками 4x4.
00048      * Масив легко інтерпретується у ‘surfaces_vector‘:
00049      *
00050      *     surfacei_t *surfaces = reinterpret_cast<surfacei_t*>(indices.data());
00051      */
00052     vector<GLuint> indices;
00053 public:
00054     /**
00055      * @brief Конструктор до Beziator, який зберігає шлях до файлу з моделлю.
00056      *
00057      * Обов’язково потібно запустити метод ‘Beziator::init‘ для того щоб
00058      * завантажити модель у пам’ять.
00059      * @param path Шлях до файлу у якому зберігається модель.
00060      */
00061     Beziator(const string& path);
00062     ~Beziator() override;
00063
00064     /**
00065      * @brief Завантажує модель у пам’ять.
00066      * @return Статус, чи успішно була завантажена модель.
00067      */
00068     bool init();
00069
00070     /**
00071      * @brief Відображує модель.
00072      *
00073      * За допомогою флагу ‘pre_generated‘ можна задати яким чином потібно
00074      * відображати, якщо задати ‘false‘, то у буде використаний меш із
00075      * поверхнями Безье 4x4, а якщо задано ‘true‘,
00076      * то відобразиться сгенерований деталізований меш моделі.
00077      * @param shader Шейдер який використовується для відображення моделі.
00078      * @param pre_generated Флаг, який позначає який з мешів відображати.
00079      */
00080     void draw(Shader &shader, bool pre_generated) override;
00081
00082     /**
00083      * @brief Генерує деталізований меш моделі.
00084      *
00085      * Ступінь деталізаії ‘d‘ позначає скільки вершин буде створено по двом
```

```
00086      * осям, за замовчанням задано 8, таким чином поверхня буде складатися з
00087      * 8x8=64 вершини.
00088      * @param d ступінь деталізації.
00089      */
00090     void generate(size_t d = 8) override;
00091
00092     /**
00093      * @brief Зберігає модель у файл, вказаний у полі ‘path‘.
00094      * @return Статус збереження файлу.
00095      */
00096     bool save();
00097
00098     /**
00099      * @brief Інвертує порядок індесів поверхні, щоб нормалі дивилися у протилежний бік.
00100      * @param i номер поверхні.
00101      */
00102     void rotate(size_t i);
00103
00104     /**
00105      * @brief Видає список вершин моделі.
00106      * @return Вказівник на поле ‘vertices‘.
00107      */
00108     vector<Vertex> *get_vertices() override;
00109 };
00110
00111 #endif // BEZIATOR_H
```

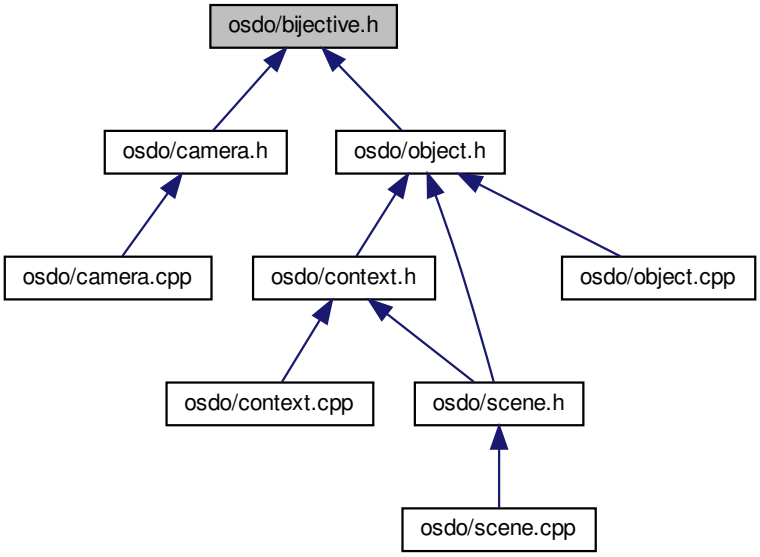
8.6 Файл osdo/bijective.h

Інтерфейс до об'єктів, що можуть можуть бути переміщені та повернуті у просторі.

#include "osdo.h"
Діаграма включених заголовочних файлів для bijective.h:



Граф файлів, які включають цей файл:



Класи

- class Bijective

  Інтерфейс до об'єктів, що можуть можуть бути переміщені та повернуті у просторі.

8.6.1 Детальний опис

Інтерфейс до об'єктів, що можуть можуть бути переміщені та повернуті у просторі.

Див. визначення в файлі bijective.h

8.7 bijective.h

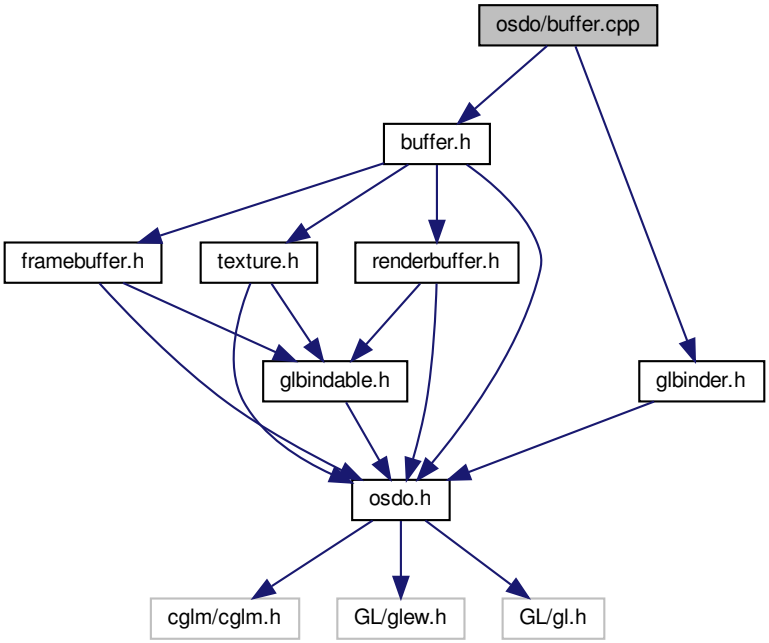
```
00001 /**
00002  * @file bijective.h
00003  * @brief Інтерфейс до об’єктів, що можуть можуть бути переміщені та повернуті у просторі.
00004  */
00005 #ifndef BIJECTIVE_H
00006 #define BIJECTIVE_H
00007
00008 #include "osdo.h"
00009
00010 /**
00011  * @brief Інтерфейс до об’єктів, що можуть можуть бути переміщені та повернуті у просторі.
00012  */
00013 class Bijective {
00014 public:
00015     virtual ~Bijective() {}
00016
00017     /**
00018      * @brief Забирає поточну позицію об’єкта у просторі.
00019      * @param[out] position поточна позицію об’єкта
00020      */
00021     virtual void get_position(vec4 position) {}
00022     /**
```

```
00023     * @brief Задає нову позицію об'єкта у просторі.
00024     * @param[in] position нова позиція об'єкта у просторі
00025     */
00026    virtual void set_position(vec4 position) {}
00027
00028    /**
00029     * @brief Забирає поточний нахил об'єкта.
00030     * @param[out] rotation поточний нахил  об'єкта
00031     */
00032    virtual void get_rotation(vec3 rotation) {}
00033    /**
00034     * @brief Задає новий нахил об'єкта.
00035     * @param[in] rotation новий нахил об'єкта
00036     */
00037    virtual void set_rotation(vec3 rotation) {}
00038
00039    /**
00040     * @brief Забирає поточну анімацію обернення об'єкта.
00041     * @param[out] rotation поточна анімація обернення об'єкта
00042     */
00043    virtual void get_animation(vec3 rotation) {}
00044    /**
00045     * @brief Задає нову анімацію обернення об'єкта.
00046     * @param[in] rotation нова анімація обернення об'єкта.
00047     */
00048    virtual void set_animation(vec3 rotation) {}
00049
00050    /**
00051     * @brief Забирає матрицю лінійних перетворень над об'єктом.
00052     * @param[out] matrix матриця лінійних перетворень
00053     */
00054    virtual void get_mat4(mat4 matrix) {}
00055
00056    /**
00057     * @brief Переміщує об'єкт у просторі.
00058     *
00059     * Переміщує об'єкт у просторі на відстані з аргументу 'distances',
00060     * де кожне значення вектору позначає відстань відповідної осі.
00061     * @param[in] distances відстані переміщення по осям
00062     * @param[in] delta_time скільки часу пройшло з останнього кадру
00063     */
00064    virtual void translate(vec3 distances, float delta_time) {}
00065    /**
00066     * @brief Обертає об'єкт.
00067     * @param[in] coord позначає координатну вісь навколо якої обертати
00068     * @param[in] delta_time скільки часу пройшло з останнього кадру
00069     */
00070    virtual void rotate(enum coord_enum coord, float delta_time) {}
00071    /**
00072     * @brief Обернути об'єкт по всім осям.
00073     * @param[in] angles вектор кутів у радіанах на кожну вісь
00074     */
00075    virtual void rotate_all(vec3 angles) {}
00076    /**
00077     * @brief Додає швидкість анімації обертання об'єкту.
00078     * @param[in] angles вектор швидкостей анімацій обертання по трьом осям
00079     * @param[in] delta_time скільки часу пройшло з останнього кадру
00080     */
00081    virtual void add_animation(vec3 angles, float delta_time) {}
00082 };
00083
00084 #endif // BIJECTIVE_H
```

8.8 Файл osdo/buffer.cpp

```
#include "buffer.h"
#include "glbinder.h"
```

Діаграма включених заголовочних файлів для buffer.cpp:



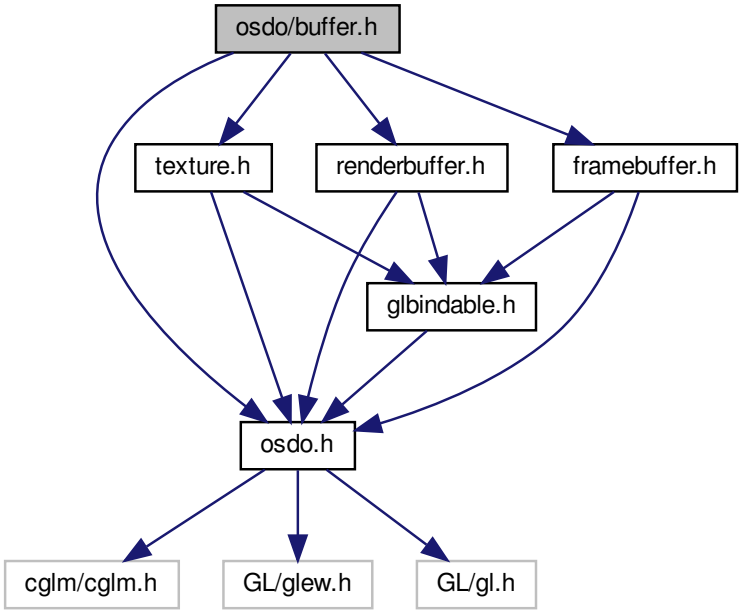
8.9 buffer.cpp

```
00001 #include "buffer.h"
00002 #include "glbinder.h"
00003
00004 //static GLenum DRAW_BUFFERS[] = {GL_COLOR_ATTACHMENT0};
00005
00006 bool Buffer::pre_render(GLsizei size[2]) {
00007     this->ms_fb.bind();
00008
00009     this->ms_fb.rb_color_multisample(size, this->color_rb);
00010     this->ms_fb.rb_depth_multisample(size, this->depth_rb);
00011
00012     if (!this->ms_fb.check()) {
00013         return false;
00014     }
00015
00016     glEnable(GL_DEPTH_TEST);
00017     glViewport(0, 0, size[0], size[1]);
00018     return true;
00019 }
00020
00021 void Buffer::post_render(GLsizei size[2]) {
00022     this->ms_fb.unbind();
00023
00024     {
00025         GlBinder b = this->fb.binder();
00026         this->fb.tex_2d(size, this->tex);
00027     }
00028     GlBinder b1 = this->ms_fb.binder(GL_READ_FRAMEBUFFER),
00029             b2 = this->fb.binder(GL_DRAW_FRAMEBUFFER);
00030
00031     glBlitFramebuffer(0, 0, size[0], size[1], 0, 0, size[0], size[1],
00032             GL_COLOR_BUFFER_BIT, GL_NEAREST);
00033 }
00034
00035 const Texture &Buffer::get_tex() {return tex;}
```

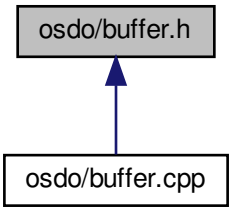
8.10 Файл osdo/buffer.h

Буфер, у якому відбувається рендеринг у текстуру.

#include "osdo.h"
#include "texture.h"
#include "renderbuffer.h"
#include "framebuffer.h"
Діаграма включених заголовочних файлів для buffer.h:



Граф файлів, які включають цей файл:



Класи

- class Buffer

  Буфер, у якому відбувається рендеринг у текстуру.

8.10.1 Детальний опис

Буфер, у якому відбувається рендеринг у текстуру.

Див. визначення в файлі buffer.h

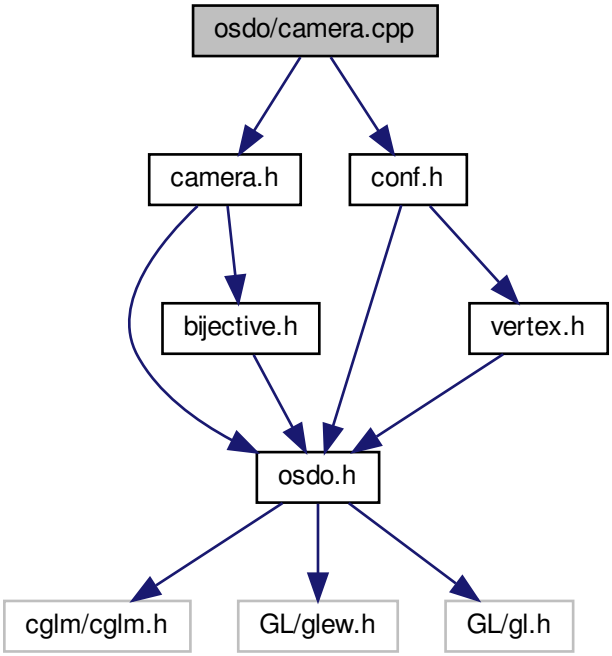
8.11 buffer.h

```
00001 /**
00002  * @file buffer.h
00003  * @brief Буфер, у якому відбувається рендеринг у текстуру.
00004  */
00005 #ifndef BUFFER_H
00006 #define BUFFER_H
00007
00008 #include "osdo.h"
00009 #include "texture.h"
00010 #include "renderbuffer.h"
00011 #include "framebuffer.h"
00012
00013 /**
00014  * @brief Буфер, у якому відбувається рендеринг у текстуру.
00015  */
00016 class Buffer {
00017     /**
00018      * @brief Текстура у яку відбувається рендеринг.
00019      */
00020     Texture tex;
00021     /**
00022      * @brief Буфер кольорів для рендеренгу.
00023      */
00024     Renderbuffer color_rb;
00025     /**
00026      * @brief Глибинний буфер для рендеренгу.
00027      */
00028     Renderbuffer depth_rb;
00029     /**
00030      * @brief Буфер утвореного кадру рендеренгу із згладжуванням.
00031      */
00032     Framebuffer ms_fb;
00033     /**
00034      * @brief Кінцевий буфер утвореного кадру.
00035      */
00036     Framebuffer fb;
00037 public:
00038     /**
00039      * @brief Підгодовка до рендеренгу.
00040      * @param[in] size ширина та висота кадру.
00041      * @return статус успішності підготовки буферів до рендеренгу.
00042      */
00043     bool pre_render(GLsizei size[2]);
00044     /**
00045      * @brief Генерація текстури з утвореного кадру.
00046      * @param[in] size ширина та висота кадру.
00047      */
00048     void post_render(GLsizei size[2]);
00049
00050     /**
00051      * @brief Забирає текстуру у яку проводився рендеринг.
00052      * @return текстура у яку проводився рендеринг
00053      */
00054     const Texture& get_tex();
00055 };
00056
00057 #endif // BUFFER_H
```

8.12 Файл osdo/camera.cpp

#include "camera.h"
#include "conf.h"

Діаграма включених заголовочних файлів для camera.cpp:



Змінні

- vec4 CAMERA_DIRECTION = CAMERA_DIRECTION_INIT

8.12.1 Опис змінних

8.12.1.1 CAMERA_DIRECTION vec4 CAMERA_DIRECTION = CAMERA_DIRECTION_INIT

Див. визначення в файлі camera.cpp, рядок 4

8.13 camera.cpp

```
00001 #include "camera.h"
00002 #include "conf.h"
00003
00004 vec4 CAMERA_DIRECTION = CAMERA_DIRECTION_INIT;
00005
00006 Camera::Camera()
00007     : rotation GLM_MAT4_IDENTITY_INIT,
00008       position GLM_VEC4_BLACK_INIT,
00009       animation GLM_VEC3_ZERO_INIT {}
00010
00011 void Camera::get_position(vec4 position)
00012 {
00013     glm_vec4_copy(this->position, position);
```

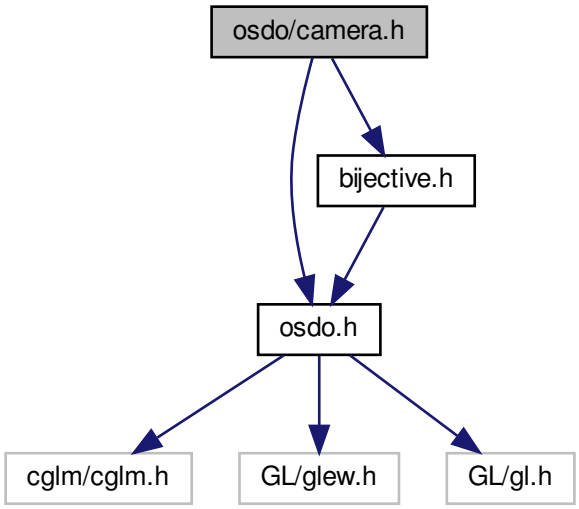
```
00014 }
00015
00016 void Camera::set_position(vec4 position)
00017 {
00018     glm_vec4_copy(position, this->position);
00019 }
00020
00021 void Camera::get_rotation(vec3 rotation)
00022 {
00023     glm_euler_angles(this->rotation, rotation);
00024 }
00025
00026 void Camera::set_rotation(vec3 rotation)
00027 {
00028     glm_euler_xyz(rotation, this->rotation);
00029 }
00030
00031 void Camera::get_animation(vec3 animation)
00032 {
00033     glm_vec3_copy(this->animation, animation);
00034 }
00035
00036 void Camera::set_animation(vec3 animation)
00037 {
00038     glm_vec3_copy(animation, this->animation);
00039 }
00040
00041 void Camera::get_mat4(mat4 dest) {
00042     Camera::get_rotation_mat4(dest);
00043     glm_translate(dest, this->position);
00044 }
00045
00046 void Camera::translate(
00047         vec3 distances, float delta_time) {
00048     vec3 new_distances = GLM_VEC3_ZERO_INIT;
00049     mat4 rotation;
00050     Camera::get_rotation_inv_mat4(rotation);
00051
00052     glm_vec3_muladds(distances, -OBJECT_MOVE_SPEED * delta_time,
00053                      new_distances);
00054     glm_vec3_rotate_m4(rotation, new_distances, new_distances);
00055     Camera::translate_camera(new_distances);
00056 }
00057
00058 void Camera::rotate(
00059         enum coord_enum coord, float delta_time) {
00060     Camera::rotate_camera(-OBJECT_ROTATE_SPEED * delta_time, coord);
00061 }
00062
00063 void Camera::rotate_all(vec3 angles) {
00064     Camera::rotate_camera(angles[0], X);
00065     Camera::rotate_camera(angles[1], Y);
00066     Camera::rotate_camera(angles[2], Z);
00067 }
00068
00069 void Camera::add_animation(
00070         vec3 angles, float delta_time) {
00071     vec3 animation;
00072     glm_vec3_muladds(angles, delta_time, animation);
00073     glm_vec3_add(this->animation, animation,
00074                  this->animation);
00075 }
00076
00077 void Camera::get_direction(vec4 dest) {
00078     mat4 matrix;
00079     Camera::get_rotation_inv_mat4(matrix);
00080     glm_mat4_mulv(matrix, CAMERA_DIRECTION, dest);
00081 }
00082
00083 void Camera::get_rotation_mat4(mat4 dest) {
00084     glm_mat4_copy(this->rotation, dest);
00085     glm_mat4_inv(dest, dest);
00086 }
00087
00088 void Camera::get_rotation_inv_mat4(mat4 dest) {
00089     glm_mat4_copy(this->rotation, dest);
00090 }
00091
00092 void Camera::translate_camera(vec3 distances) {
00093     glm_vec3_add(this->position, distances, this->position);
00094 }
00095
00096 void Camera::rotate_camera(float angle, enum coord_enum coord) {
00097     switch (coord) {
00098     case X: glm_rotate_x(this->rotation, angle, this->rotation); break;
00099     case Y: glm_rotate_y(this->rotation, angle, this->rotation); break;
00100     case Z: glm_rotate_z(this->rotation, angle, this->rotation); break;
```

```
00101     }
00102     /*if (glm_vec3_dot(this->rotation[2], GLM_XUP) > 0.1f) {
00103         glm_cross(this->rotation[2], GLM_XUP, this->rotation[1]);
00104         glm_cross(this->rotation[1], this->rotation[2], this->rotation[0]);
00105         glm_normalize(this->rotation[0]);
00106         glm_normalize(this->rotation[1]);
00107     }*/
00108 }
00109
00110 void Camera::rotate_all_camera(vec3 angles) {
00111     Camera::rotate_all(angles);
00112 }
00113
00114 void Camera::rotate_all_inverse(vec3 angles) {
00115     mat4 m = GLM_MAT4_IDENTITY_INIT;
00116     vec4 v = GLM_VEC4_BLACK_INIT;
00117
00118     glm_vec3_copy(rotation[0], v);
00119     glm_rotate(m, angles[0], v);
00120     glm_vec3_copy(rotation[1], v);
00121     glm_rotate(m, angles[1], v);
00122     glm_vec3_copy(rotation[2], v);
00123     glm_rotate(m, angles[2], v);
00124     glm_mat4_mul(m, rotation, rotation);
00125     glm_mat4_mulv(m, position, position);
00126 }
```

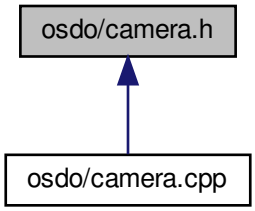
8.14 Файл osdo/camera.h

Клас камери, якою можна маніпулювати у сцені.

#include "osdo.h"
#include "bijective.h"
Діаграма включених заголовочних файлів для camera.h:



Граф файлів, які включають цей файл:



Класи

- class Camera

  Клас камери, якою можна маніпулювати у сцені.

Макровизначення

- #define CAMERA_DIRECTION_INIT {0.0f, 0.0f, -1.0f, 0.0f}

  Вектор, який використовуэться для обчислення напрямку перегляду камери.

8.14.1 Детальний опис

Клас камери, якою можна маніпулювати у сцені.

Див. визначення в файлі camera.h

8.14.2 Опис макровизначень

8.14.2.1 CAMERA_DIRECTION_INIT #define CAMERA_DIRECTION_INIT {0.0f, 0.0f, -1.0f, 0.0f}

Вектор, який використовуэться для обчислення напрямку перегляду камери.

Див. визначення в файлі camera.h, рядок 14

8.15 camera.h

```
00001 /**
00002  * @file camera.h
00003  * @brief Клас камери, якою можна маніпулювати у сцені.
00004  */
00005 #ifndef CAMERA_H
00006 #define CAMERA_H
00007
00008 #include "osdo.h"
00009 #include "bijective.h"
00010
00011 /**
00012  * @brief Вектор, який використовуэться для обчислення напрямку перегляду камери.
00013  */
00014 #define CAMERA_DIRECTION_INIT {0.0f, 0.0f, -1.0f, 0.0f}
00015
00016 /**
00017  * @brief Клас камери, якою можна маніпулювати у сцені.
00018  */
00019 class Camera : public Bijective {
00020     /**
00021      * @brief Матриця лінійних перетворень у якій зберігається обернення камери.
00022      */
00023     mat4 rotation;
00024     /**
00025      * @brief Вектор позіції камери \f$(x, y, z, 1.0)\f$
00026      */
00027     vec4 position;
00028     /**
00029      * @brief Вектор, який реалізования для інтерфейсу ‘Bijective‘
00030      */
00031     vec4 animation;
00032 public:
00033     Camera();
00034
00035     /**
00036      * @brief Забирає поточну позицію камери у просторі.
00037      * @param[out] position поточна позицію камери
00038      */
00039     void get_position(vec4 position) override;
00040     /**
00041      * @brief Задає нову позицію камери у просторі.
00042      * @param[in] position нова позиція камери у просторі
00043      */
00044     void set_position(vec4 position) override;
00045
00046     /**
00047      * @brief Забирає поточний нахил камери.
00048      * @param[out] rotation поточний нахил камери
00049      */
00050     void get_rotation(vec3 rotation) override;
00051     /**
00052      * @brief Задає новий нахил камери.
00053      * @param[in] rotation новий нахил камери
00054      */
00055     void set_rotation(vec3 rotation) override;
00056
00057     /**
00058      * @brief Метод реалізований для інтерфейсу ‘Bijective‘
00059      */
00060     void get_animation(vec3 animation) override;
00061     /**
00062      * @brief Метод реалізований для інтерфейсу ‘Bijective‘
00063      */
00064     void set_animation(vec3 animation) override;
00065
00066     /**
00067      * @brief Забирає матрицю лінійних перетворень над камерою.
00068      * @param[out] matrix матриця лінійних перетворень
00069      */
00070     void get_mat4(mat4 matrix) override;
00071
00072     /**
00073      * @brief Переміщує камеру у просторі.
00074      * @param[in] distances відстані переміщення по осям
00075      * @param[in] delta_time скільки часу пройшло з останнього кадру
00076      */
00077     void translate(vec3 distances, float delta_time) override;
00078     /**
00079      * @brief Обертає камеру.
00080      * @param[in] coord позначає координатну вісь навколо якої обертати
00081      * @param[in] delta_time скільки часу пройшло з останнього кадру
00082      */
00083     void rotate(enum coord_enum coord, float delta_time) override;
00084     /**
00085      * @brief Обернути камеру по всім осям.
```

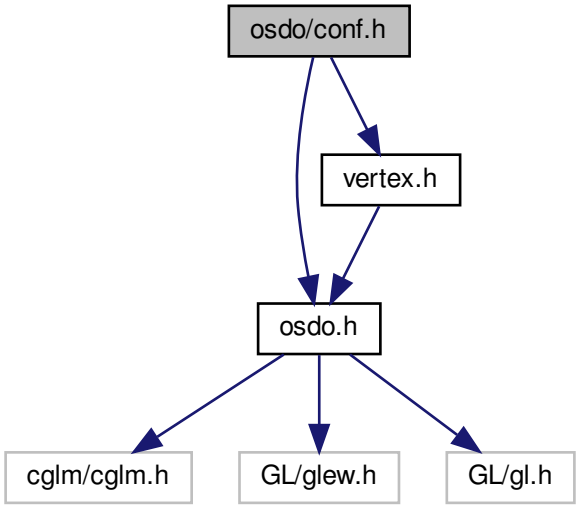
```
00086     * @param[in] angles вектор кутів у радіанах на кожну вісь
00087     */
00088    void rotate_all(vec3 angles) override;
00089    /**
00090     * @brief Метод реалізований для інтерфейсу 'Bijective'
00091     */
00092    void add_animation(vec3 angles, float delta_time) override;
00093
00094    /**
00095     * @brief Обчислює напрямок перегляду камери.
00096     * @param[out] dest напрямок перегляду камери
00097     */
00098    void get_direction(vec4 dest);
00099    /**
00100     * @brief Забирає матрицю обертання камери.
00101     * @param[out] dest матриця обертання камери.
00102     */
00103    void get_rotation_mat4(mat4 dest);
00104    /**
00105     * @brief Забирає інвертовану матрицю обертання камери.
00106     * @param[out] dest інвертована матриця обертання камери.
00107     */
00108    void get_rotation_inv_mat4(mat4 dest);
00109
00110    /**
00111     * @brief Переміщує камеру незалежно від часу.
00112     * @param distances відстані переміщення по трьом осям \f$(x, y, z)\f$
00113     */
00114    void translate_camera(vec3 distances);
00115
00116    /**
00117     * @brief Обертає камеру по осі незалежно від часу.
00118     * @param angle кут обертання у радіанах
00119     * @param coord ввісь обертання
00120     */
00121    void rotate_camera(float angle, enum coord_enum coord);
00122    /**
00123     * @brief Обертає камеру по всім осям незалежно від часу.
00124     * @param angles вектор кутів обертання по трьом осям \f$(x, y, z)\f$
00125     */
00126    void rotate_all_camera(vec3 angles);
00127
00128    /**
00129     * @brief Обертає камеру по орбіті навколо центру коодинат.
00130     * @param angles вектор кутів у радіанах обертання по трьом осям \f$(x, y, z)\f$
00131     */
00132    void rotate_all_inverse(vec3 angles);
00133
00134 };
00135
00136 #endif // CAMERA_H
```

8.16 Файл osdo/conf.h

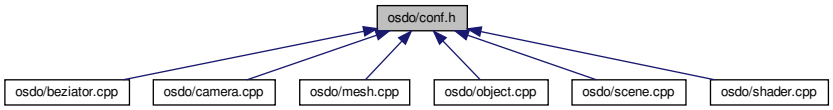
Конфігурація бібліотеки osdo

#include "osdo.h"
#include "vertex.h"

Діаграма включених заголовочних файлів для conf.h:



Граф файлів, які включають цей файл:



Макровизначення

- #define M_PI 3.14159265358979323846
  Число пі у типі double
- #define M_RAD M_PI / 180
  Коеффіцієнт перетворення у радіани у типі double
- #define M_PI_F 3.14159265358979323846f
- #define M_RAD_F M_PI_F / 180
  Число пі у типі float
- #define RES_DIR "../share/osdo"
  Коеффіцієнт перетворення у радіани у типі float
- #define VERTEX_PATH RES_DIR"/%s.vert"
  Паттерн шляху до файлу з вершинним шейдером
- #define TESC_PATH RES_DIR"/%s.tesc"
  Паттерн шляху до файлу з теселяційним контрольним шейдером
- #define TESE_PATH RES_DIR"/%s.tese"
  Паттерн шляху до файлу з теселяційним обчислювальним шейдером

- #define GEOMETRY_PATH RES_DIR"/%s.geom"

  Паттерн шляху до файлу з геометричним шейдером
- #define FRAGMENT_PATH RES_DIR"/%s.frag"

  Паттерн шляху до файлу з фрагментним шейдером
- #define BEZIATOR_PATH RES_DIR"/%s.odom"

  Паттерн шляху до файлу з моделлю
- #define MAX_VERTEX_BUFFER 512 * 1024

  Максимальна кількість вершин
- #define MAX_ELEMENT_BUFFER 128 * 1024

  Максимальна кількість елементів
- #define NK_GLFW_DOUBLE_CLICK_LO 0.02

  Нижня границя часу при якому спрацьовує подвійний клік мишою
- #define NK_GLFW_DOUBLE_CLICK_HI 0.2

  Верхня границя часу при якому спрацьовує подвійний клік мишою

8.16.1 Детальний опис

Конфігурація бібліотеки osdo

Див. визначення в файлі conf.h

8.16.2 Опис макровизначень

8.16.2.1 BEZIATOR_PATH #define BEZIATOR_PATH RES_DIR"/%s.odom"

Паттерн шляху до файлу з моделлю

Див. визначення в файлі conf.h, рядок 56

8.16.2.2 FRAGMENT_PATH #define FRAGMENT_PATH RES_DIR"/%s.frag"

Паттерн шляху до файлу з фрагментним шейдером

Див. визначення в файлі conf.h, рядок 52

8.16.2.3 GEOMETRY_PATH #define GEOMETRY_PATH RES_DIR"/%s.geom"

Паттерн шляху до файлу з геометричним шейдером

Див. визначення в файлі conf.h, рядок 48

8.16.2.4 M_PI #define M_PI 3.14159265358979323846

Число пі у типі double

Див. визначення в файлі conf.h, рядок 14

8.16.2.5 M_PI_F #define M_PI_F 3.14159265358979323846f

Див. визначення в файлі conf.h, рядок 19

8.16.2.6 M_RAD #define M_RAD M_PI / 180

Коеффіцієнт перетворення у радіани у типі double

Див. визначення в файлі conf.h, рядок 18

8.16.2.7 M_RAD_F #define M_RAD_F M_PI_F / 180

Число пі у типі float

Див. визначення в файлі conf.h, рядок 23

8.16.2.8 MAX_ELEMENT_BUFFER #define MAX_ELEMENT_BUFFER 128 * 1024

Максимальна кількість елементів

Див. визначення в файлі conf.h, рядок 65

8.16.2.9 MAX_VERTEX_BUFFER #define MAX_VERTEX_BUFFER 512 * 1024

Максимальна кількість вершин

Див. визначення в файлі conf.h, рядок 61

8.16.2.10 NK_GLFW_DOUBLE_CLICK_HI #define NK_GLFW_DOUBLE_CLICK_HI 0.2

Верхня границя часу при якому спрацьовує подвійний клік мишою

Див. визначення в файлі conf.h, рядок 74

8.16.2.11 NK_GLFW_DOUBLE_CLICK_LO #define NK_GLFW_DOUBLE_CLICK_LO 0.02

Нижня границя часу при якому спрацьовує подвійний клік мишою

Див. визначення в файлі conf.h, рядок 70

8.16.2.12 RES_DIR #define RES_DIR "../share/osdo"

Коеффіцієнт перетворення у радіани у типі float

Шлях до директорії з ресурсами бібліотеки osdo

Див. визначення в файлі conf.h, рядок 31

8.16.2.13 TESC_PATH #define TESC_PATH RES_DIR"/%s.tesc"

Паттерн шляху до файлу з теселяційним контрольним шейдером

Див. визначення в файлі conf.h, рядок 40

8.16.2.14 TESE_PATH #define TESE_PATH RES_DIR"/%s.tese"

Паттерн шляху до файлу з теселяційним обчислювальним шейдером

Див. визначення в файлі conf.h, рядок 44

8.16.2.15 VERTEX_PATH #define VERTEX_PATH RES_DIR"/%s.vert"

Паттерн шляху до файлу з вершинним шейдером

Див. визначення в файлі conf.h, рядок 36

8.17 conf.h

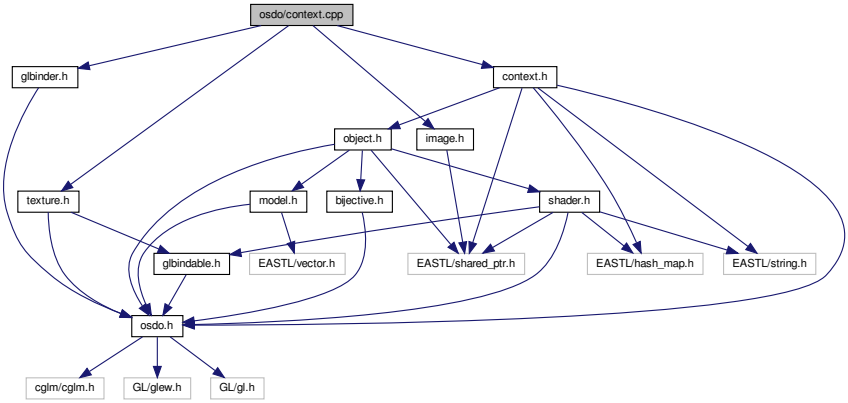
```
00001 /**
00002  * @file conf.h
00003  * @brief Конфігурація бібліотеки ‘osdo‘
00004  */
00005 #ifndef CONF_H
00006 #define CONF_H
00007
00008 #include "osdo.h"
00009 #include "vertex.h"
00010
00011 /**
00012  * @brief Число пі у типі ‘double‘
00013  */
00014 #define M_PI 3.14159265358979323846
00015 /**
00016  * @brief Коеффіцієнт перетворення у радіани у типі ‘double‘
00017  */
00018 #define M_RAD M_PI / 180
00019 #define M_PI_F 3.14159265358979323846f
00020 /**
00021  * @brief Число пі у типі ‘float‘
00022  */
00023 #define M_RAD_F M_PI_F / 180
00024 /**
00025  * @brief Коеффіцієнт перетворення у радіани у типі ‘float‘
00026  */
00027
00028 /**
00029  * @brief Шлях до директорії з ресурсами бібліотеки ‘osdo‘
00030  */
00031 #define RES_DIR "../share/osdo"
00032
00033 /**
00034  * @brief Паттерн шляху до файлу з вершинним шейдером
00035  */
00036 #define VERTEX_PATH RES_DIR"/%s.vert"
00037 /**
00038  * @brief Паттерн шляху до файлу з теселяційним контрольним шейдером
00039  */
00040 #define TESC_PATH RES_DIR"/%s.tesc"
00041 /**
00042  * @brief Паттерн шляху до файлу з теселяційним обчислювальним шейдером
00043  */
00044 #define TESE_PATH RES_DIR"/%s.tese"
00045 /**
00046  * @brief Паттерн шляху до файлу з геометричним шейдером
00047  */
00048 #define GEOMETRY_PATH RES_DIR"/%s.geom"
00049 /**
00050  * @brief Паттерн шляху до файлу з фрагментним шейдером
00051  */
00052 #define FRAGMENT_PATH RES_DIR"/%s.frag"
00053 /**
00054  * @brief Паттерн шляху до файлу з моделлю
00055  */
00056 #define BEZIATOR_PATH RES_DIR"/%s.odom"
00057
00058 /**
00059  * @brief Максимальна кількість вершин
00060  */
00061 #define MAX_VERTEX_BUFFER 512 * 1024
00062 /**
00063  * @brief Максимальна кількість елементів
00064  */
00065 #define MAX_ELEMENT_BUFFER 128 * 1024
00066
00067 /**
00068  * @brief Нижня границя часу при якому спрацьовує подвійний клік мишою
00069  */
00070 #define NK_GLFW_DOUBLE_CLICK_LO 0.02
00071 /**
00072  * @brief Верхня границя часу при якому спрацьовує подвійний клік мишою
00073  */
00074 #define NK_GLFW_DOUBLE_CLICK_HI 0.2
00075
00076 /**
00077  * @brief Початкова ширина вікна при запуску програми
00078  */
00079 static const unsigned int SCR_WIDTH = 1366;
00080 /**
00081  * @brief Початкова висота вікна при запуску програми
00082  */
00083 static const unsigned int SCR_HEIGHT = 700;
00084
00085 /**
```

```
00086  * @brief Швидкість переміщення об'єкту
00087  */
00088 static const float OBJECT_MOVE_SPEED   =  5.0f;
00089 /**
00090  * @brief Швидкість повороту об'єкту
00091  */
00092 static const float OBJECT_ROTATE_SPEED =  1.0f;
00093 /**
00094  * @brief Коєффіцієнти швидкості для додавання обертання об'єкту
00095  */
00096 static const float OBJECT_ANIMATE_SPEED = 1.0f;
00097
00098 /**
00099  * @brief Чутливість миші
00100  */
00101 static const float SENSITIVITY = 0.01f;
00102
00103 /**
00104  * @brief Початкові позиції джерел світла (не використовується у програмі)
00105  */
00106 static vec3 UNUSED LAMP_POSITIONS[] = {
00107     {5.0f, 0.0f, 5.0f},
00108     {-1.0f, 0.0f, 1.0f}
00109 };
00110
00111 /**
00112  * @brief Вершини куба
00113  */
00114 static const Vertex EXAMPLE_CUBE_VERTEX[] = {
00115     {{-1.,  1., -1.}, { 0.,  1.,  0.}, {  0, 255,   0, 255}, {0., 0.}},
00116     {{ 1.,  1.,  1.}, { 0.,  1.,  0.}, {255, 255, 255, 255}, {0., 0.}},
00117     {{ 1.,  1., -1.}, { 0.,  1.,  0.}, {255, 255,   0, 255}, {0., 0.}},
00118
00119     {{ 1.,  1.,  1.}, { 0.,  0.,  1.}, {255, 255, 255, 255}, {0., 0.}},
00120     {{-1., -1.,  1.}, { 0.,  0.,  1.}, {  0,   0, 255, 255}, {0., 0.}},
00121     {{ 1., -1.,  1.}, { 0.,  0.,  1.}, {255,   0, 255, 255}, {0., 0.}},
00122
00123     {{-1.,  1.,  1.}, {-1.,  0.,  0.}, {  0, 255, 255, 255}, {0., 0.}},
00124     {{-1., -1., -1.}, {-1.,  0.,  0.}, {  0,   0,   0, 255}, {0., 0.}},
00125     {{-1., -1.,  1.}, {-1.,  0.,  0.}, {  0,   0, 255, 255}, {0., 0.}},
00126
00127     {{ 1., -1., -1.}, { 0., -1.,  0.}, {255,   0,   0, 255}, {0., 0.}},
00128     {{-1., -1.,  1.}, { 0., -1.,  0.}, {  0,   0, 255, 255}, {0., 0.}},
00129     {{-1., -1., -1.}, { 0., -1.,  0.}, {  0,   0,   0, 255}, {0., 0.}},
00130
00131     {{ 1.,  1., -1.}, { 1.,  0.,  0.}, {255, 255,   0, 255}, {0., 0.}},
00132     {{ 1., -1.,  1.}, { 1.,  0.,  0.}, {255,   0, 255, 255}, {0., 0.}},
00133     {{ 1., -1., -1.}, { 1.,  0.,  0.}, {255,   0,   0, 255}, {0., 0.}},
00134
00135     {{-1.,  1., -1.}, { 0.,  0., -1.}, {  0, 255,   0, 255}, {0., 0.}},
00136     {{ 1., -1., -1.}, { 0.,  0., -1.}, {255,   0,   0, 255}, {0., 0.}},
00137     {{-1., -1., -1.}, { 0.,  0., -1.}, {  0,   0,   0, 255}, {0., 0.}},
00138
00139     {{-1.,  1., -1.}, { 0.,  1.,  0.}, {  0, 255,   0, 255}, {0., 0.}},
00140     {{-1.,  1.,  1.}, { 0.,  1.,  0.}, {  0, 255, 255, 255}, {0., 0.}},
00141     {{ 1.,  1.,  1.}, { 0.,  1.,  0.}, {255, 255, 255, 255}, {0., 0.}},
00142
00143     {{ 1.,  1.,  1.}, { 0.,  0.,  1.}, {255, 255, 255, 255}, {0., 0.}},
00144     {{-1.,  1.,  1.}, { 0.,  0.,  1.}, {  0, 255, 255, 255}, {0., 0.}},
00145     {{-1., -1.,  1.}, { 0.,  0.,  1.}, {  0,   0, 255, 255}, {0., 0.}},
00146
00147     {{-1.,  1.,  1.}, {-1.,  0.,  0.}, {  0, 255, 255, 255}, {0., 0.}},
00148     {{-1.,  1., -1.}, {-1.,  0.,  0.}, {  0, 255,   0, 255}, {0., 0.}},
00149     {{-1., -1., -1.}, {-1.,  0.,  0.}, {  0,   0,   0, 255}, {0., 0.}},
00150
00151     {{ 1., -1., -1.}, { 0., -1.,  0.}, {255,   0,   0, 255}, {0., 0.}},
00152     {{ 1., -1.,  1.}, { 0., -1.,  0.}, {255,   0, 255, 255}, {0., 0.}},
00153     {{-1., -1.,  1.}, { 0., -1.,  0.}, {  0,   0, 255, 255}, {0., 0.}},
00154
00155     {{ 1.,  1., -1.}, { 1.,  0.,  0.}, {255, 255,   0, 255}, {0., 0.}},
00156     {{ 1.,  1.,  1.}, { 1.,  0.,  0.}, {255, 255, 255, 255}, {0., 0.}},
00157     {{ 1., -1.,  1.}, { 1.,  0.,  0.}, {255,   0, 255, 255}, {0., 0.}},
00158
00159     {{-1.,  1., -1.}, { 0.,  0., -1.}, {  0, 255,   0, 255}, {0., 0.}},
00160     {{ 1.,  1., -1.}, { 0.,  0., -1.}, {255, 255,   0, 255}, {0., 0.}},
00161     {{ 1., -1., -1.}, { 0.,  0., -1.}, {255,   0,   0, 255}, {0., 0.}},
00162 };
00163
00164 /**
00165  * @brief Індекси вершин, що утворюють сторони куба.
00166  */
00167 static const GLuint EXAMPLE_CUBE_INDICIES[] = {
00168      0,  1,  2,  3,  4,  5,  6,  7,  8,  9, 10, 11,
00169     12, 13, 14, 15, 16, 17, 18, 19, 20, 21, 22, 23,
00170     24, 25, 26, 27, 28, 29, 30, 31, 32, 33, 34, 35,
00171 };
00172
```

00173 #endif // CONF_H

8.18 Файл osdo/context.cpp

#include "context.h"
#include "glbinder.h"
#include "image.h"
#include "texture.h"
Діаграма включених заголовочних файлів для context.cpp:



8.19 context.cpp

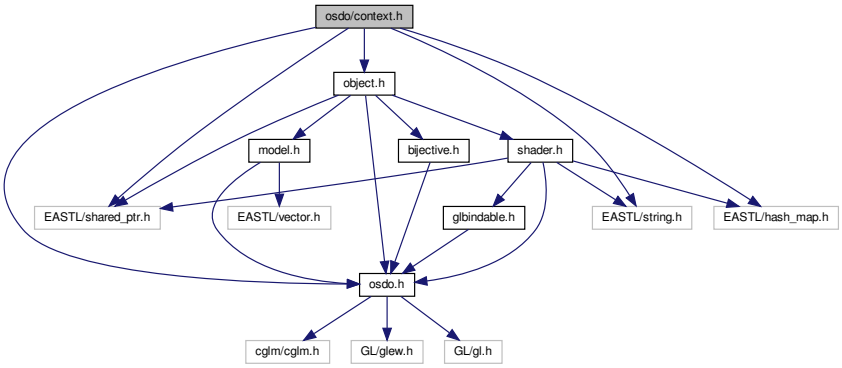
```
00001 #include "context.h"
00002 #include "glbinder.h"
00003 #include "image.h"
00004 #include "texture.h"
00005
00006 Context::Context() : active(models.end()), active_texture(textures.end()) {
00007
00008 }
00009
00010 Context::Models::iterator &Context::next_active() {
00011     if (active == models.end()) {
00012         active = models.begin();
00013     } else active++;
00014     return active;
00015 }
00016
00017 void Context::load_texture(const char *path) {
00018     Image img = Image::fromFile(path);
00019     if (img.data) {
00020         auto tex = make_shared<Texture>();
00021         tex->update(img);
00022         textures[path] = tex;
00023     }
00024 }
00025
00026 bool Context::load_shader(const char *name, const Shader::shader_map& shaders) {
00027     auto shader = Shader::create(shaders);
00028     if (!shader)
00029         return false;
00030     this->shaders[string(name)] = shader;
00031     return true;
00032 }
```

8.20 Файл osdo/context.h

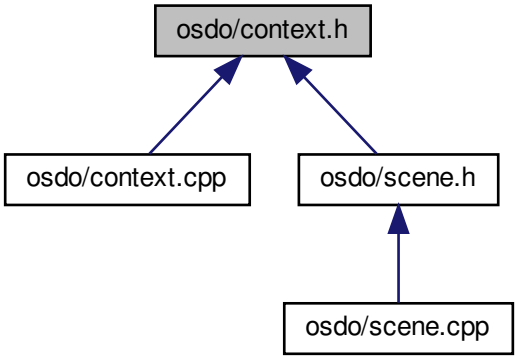
Контекст, який зберігає усі завантажені у пам'ять ресурси.

```
#include "osdo.h"
#include "object.h"
#include "EASTL/hash_map.h"
#include "EASTL/string.h"
#include "EASTL/shared_ptr.h"
```

Діаграма включених заголовочних файлів для context.h:



Граф файлів, які включають цей файл:



Класи

- struct Context

  Контекст, який зберігає усі завантажені у пам'ять ресурси.

8.20.1 Детальний опис

Контекст, який зберігає усі завантажені у пам'ять ресурси.

Див. визначення в файлі context.h

8.21 context.h

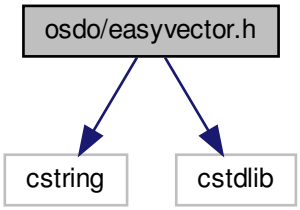
```
00001 /**
00002  * @file context.h
00003  * @brief Контекст, який зберігає усі завантажені у пам'ять ресурси.
00004  */
00005 #ifndef CONTEXT_H
00006 #define CONTEXT_H
00007
00008 #include "osdo.h"
00009
00010 #include "object.h"
00011 #include "EASTL/hash_map.h"
00012 #include "EASTL/string.h"
00013 #include "EASTL/shared_ptr.h"
00014 using eastl::hash_map;
00015 using eastl::string;
00016 using eastl::shared_ptr;
00017 using eastl::pair;
00018 using eastl::make_shared;
00019
00020 class Shader;
00021 class Texture;
00022
00023 /**
00024  * @brief Контекст, який зберігає усі завантажені у пам'ять ресурси.
00025  */
00026 struct Context
00027 {
00028     /**
00029      * @brief Тип для зберігання моделей.
00030      */
00031     typedef hash_map<string, Object> Models;
00032     /**
00033      * @brief Тип для зберігання текстур.
00034      */
00035     typedef hash_map<string, shared_ptr<Texture» Textures;
00036     /**
00037      * @brief Завантажені моделі.
00038      */
00039     Models models;
00040     /**
00041      * @brief Зкомпіловані шейдери.
00042      */
00043     hash_map<string, shared_ptr<Shader» shaders;
00044     /**
00045      * @brief Завантажені текстури.
00046      */
00047     Textures textures;
00048
00049     /**
00050      * @brief Вибрана модель для редагування.
00051      */
00052     Models::iterator active;
00053     /**
00054      * @brief Вибрана текстура для відображення.
00055      */
00056     Textures::iterator active_texture;
00057
00058 public:
00059     Context();
00060
00061     /**
00062      * @brief перехід до наступної моделі для редагування.
00063      * @return ітератор моделі
00064      */
00065     Models::iterator &next_active();
00066
00067     /**
00068      * @brief Завантажує текстуру у пам'ять.
00069      * @param[in] path шлях до файлу з текстурою
00070      */
00071     void load_texture(const char *path);
00072
00073     /**
00074      * @brief Завантажує та компілює шейдер.
00075      * @param name назва шейдеру
00076      * @param shaders массив до файлів шейдеру
00077      * @return статус успішності завантаження та компіляції шейдеру
00078      */
00079     bool load_shader(const char *name, const Shader::shader_map& shaders);
00080
00081     /**
00082      * @brief Завантажує модель з поверхнями Безье
00083      * @param path шлях до файлу з моделлю
00084      * @return статус успішності завантаження моделі
00085      */
```

```
00086     bool load_model(const string& path);
00087 };
00088
00089 #endif // CONTEXT_H
```

8.22 Файл osdo/easyvector.h

Вектор що не змінює свій розмір.

#include <cstring>
#include <cstdlib>
Діаграма включених заголовочних файлів для easyvector.h:



Класи

- class OSDO::vector< T >
  Вектор що не змінює свій розмір.

Простори імен

- OSDO
  Неймспейс бібліотеки osdo

8.22.1 Детальний опис

Вектор що не змінює свій розмір.

Див. визначення в файлі easyvector.h

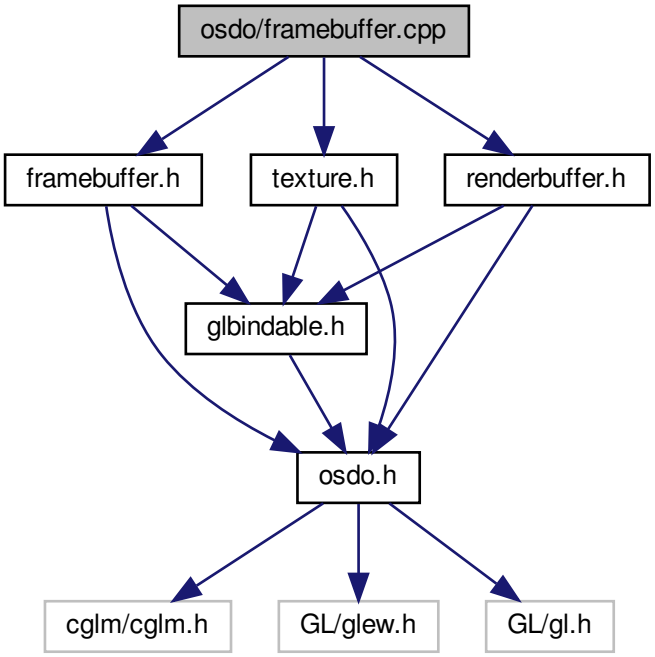
8.23 easyvector.h

```
00001 /**
00002  * @file easyvector.h
00003  * @brief Вектор що не змінює свій розмір.
00004  */
00005 #ifndef EASYVECTOR_H
00006 #define EASYVECTOR_H
00007
00008 #include <cstring>
00009 #include <cstdlib>
00010
00011 /**
00012  * @brief Неймспейс бібліотеки ‘osdo‘
00013  */
00014 namespace OSDO {
00015 /**
00016  * @brief Вектор що не змінює свій розмір.
00017  */
00018     template<class T>
00019     class vector {
00020         /**
00021          * @brief Масив із елементами типу ‘T‘
00022          */
00023         T * arr;
00024         /**
00025          * @brief Поточний розмір масиву.
00026          */
00027         size_t _size;
00028         /**
00029          * @brief Виділяє масив розміру ‘size‘
00030          * @param size розмір масиву
00031          */
00032         void _allocate(size_t size) {
00033             _size = 0;
00034             arr = nullptr;
00035             if (size)
00036                 arr = static_cast<T*>(calloc(size, sizeof(T)));
00037             if (arr)
00038                 _size = size;
00039         }
00040         /**
00041          * @brief Звільнює пам’ять від масиву.
00042          */
00043         void _free() {
00044             if (arr)
00045                 free(arr);
00046             _size = 0;
00047             arr = nullptr;
00048         }
00049         /**
00050          * @brief Копіює масив.
00051          * @param vector інший масив
00052          */
00053         void _copy(const vector& vector) {
00054             _allocate(vector._size);
00055             if (_size)
00056                 memcpy(arr, vector.arr, _size * sizeof(T));
00057         }
00058         /**
00059          * @brief Переміщює данні з іншого масиву.
00060          * @param vector інший масив
00061          */
00062         void _move(vector& vector) {
00063             arr = vector.arr;
00064             _size = vector._size;
00065         }
00066     public:
00067         /**
00068          * @brief Конструктор з параметром розміру масиву
00069          * @param size розмір массиву
00070          */
00071         vector(size_t size = 0) {
00072             _allocate(size);
00073         }
00074         /**
00075          * @brief Конструктор переносу
00076          * @param vector інший масив
00077          */
00078         vector(vector&& vector) {
00079             _copy(vector);
00080         }
00081         /**
00082          * @brief Конструктор копіювання
00083          * @param vector інший масив
00084          */
00085         vector(const vector& vector) {
```

```
00086             _copy(vector);
00087         }
00088         ~vector() {_free();}
00089         vector &operator=(vector&& vector) {
00090             _free();
00091             _copy(vector);
00092             return *this;
00093         }
00094         vector &operator=(const vector& vector) {
00095             _free();
00096             _copy(vector);
00097             return *this;
00098         }
00099         T &operator[](size_t i) {
00100             return arr[i];
00101         }
00102         size_t size() const {
00103             return _size;
00104         }
00105         T * data() {
00106             return arr;
00107         }
00108         const T * data() const {
00109             return arr;
00110         }
00111         void clear() {
00112             *this = vector(0);
00113         }
00114     };
00115 }
00116
00117 #endif // EASYVECTOR_H
```

8.24 Файл osdo/framebuffer.cpp

#include "framebuffer.h"
#include "texture.h"
#include "renderbuffer.h"
Діаграма включених заголовочних файлів для framebuffer.cpp:



8.25 framebuffer.cpp

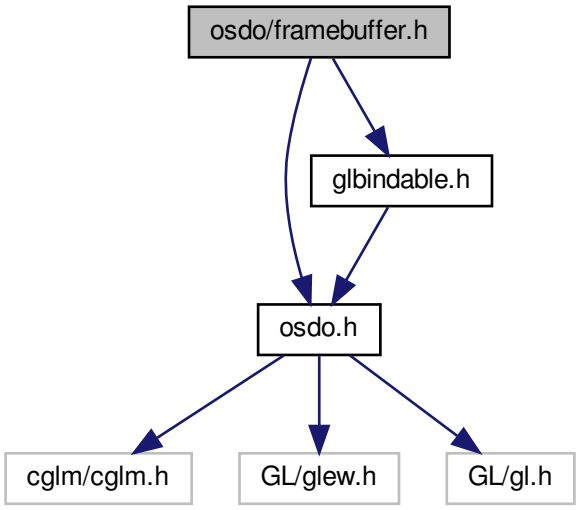
```
00001 #include "framebuffer.h"
00002 #include "texture.h"
00003 #include "renderbuffer.h"
00004
00005 GLuint Framebuffer::_generate() const
00006 {
00007     GLuint id;
00008     glGenFramebuffers(1, &id);
00009     return id;
00010 }
00011
00012 void Framebuffer::_bind(const GLuint id, GLenum target) const
00013 {
00014     glBindFramebuffer(target, id);
00015 }
00016
00017 GLenum Framebuffer::_default() const
00018 {
00019     return GL_FRAMEBUFFER;
00020 }
00021
00022 Framebuffer::Framebuffer() : GlBindable(_generate()) {}
00023
00024 Framebuffer::~Framebuffer() {
00025     glDeleteFramebuffers(1, &get_id());
00026 }
00027
00028 bool Framebuffer::check(GLenum target) {
00029     return glCheckFramebufferStatus(target) == GL_FRAMEBUFFER_COMPLETE;
00030 }
00031
00032 void Framebuffer::tex_2d_multisample(GLsizei size[2], const Texture &texture) {
00033     texture.make_2d_multisample(size);
00034     glFramebufferTexture2D(GL_FRAMEBUFFER, GL_COLOR_ATTACHMENT0,
00035                            GL_TEXTURE_2D_MULTISAMPLE, texture.get_id(), 0);
00036 }
00037
00038 void Framebuffer::tex_2d(GLsizei size[2], const Texture &texture) {
00039     texture.make_2d(size);
00040     glFramebufferTexture(GL_FRAMEBUFFER, GL_COLOR_ATTACHMENT0,
00041                          texture.get_id(), 0);
00042 }
00043
00044 void Framebuffer::renderbuffer(const Renderbuffer &rb, GLenum target) const {
00045     glFramebufferRenderbuffer(GL_FRAMEBUFFER, target,
00046                               GL_RENDERBUFFER, rb.get_id());
00047 }
00048
00049 void Framebuffer::rb_color_multisample(
00050         GLsizei size[2], const Renderbuffer &rb) const {
00051     rb.make_multisample(size, GL_RGB);
00052     renderbuffer(rb, GL_COLOR_ATTACHMENT0);
00053 }
00054
00055 void Framebuffer::rb_depth_multisample(
00056         GLsizei size[2], const Renderbuffer &rb) const {
00057     rb.make_multisample(size, GL_DEPTH32F_STENCIL8);
00058     renderbuffer(rb, GL_DEPTH_STENCIL_ATTACHMENT);
00059 }
00060
00061 void Framebuffer::rb_color(GLsizei size[2], const Renderbuffer &rb) const {
00062     rb.make(size, GL_RGB);
00063     renderbuffer(rb, GL_COLOR_ATTACHMENT0);
00064 }
00065
00066
00067 void Framebuffer::rb_depth(GLsizei size[2], const Renderbuffer &rb) const {
00068     rb.make(size, GL_DEPTH32F_STENCIL8);
00069     renderbuffer(rb, GL_DEPTH_STENCIL_ATTACHMENT);
00070 }
```

8.26 Файл osdo/framebuffer.h

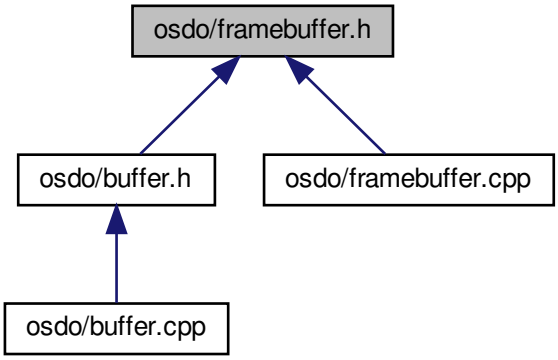
Буфер кадру, що використовується для рендеренгу.

#include "osdo.h"
#include "glbindable.h"

Діаграма включених заголовочних файлів для framebuffer.h:



Граф файлів, які включають цей файл:



Класи

- class Framebuffer

  Буфер кадру, що використовується для рендеренгу.

8.26.1 Детальний опис

Буфер кадру, що використовується для рендеренгу.

Див. визначення в файлі framebuffer.h

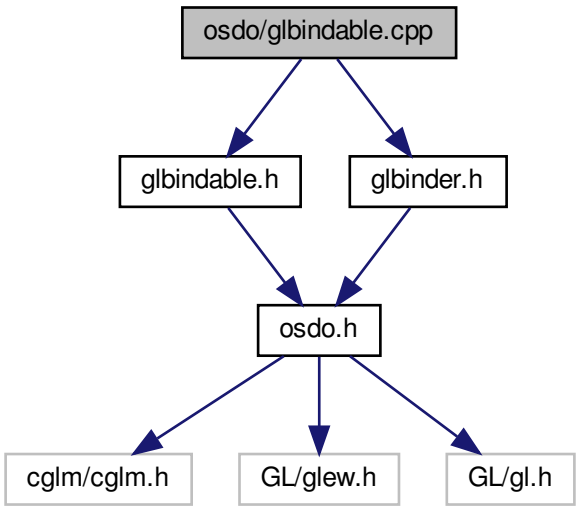
8.27 framebuffer.h

```
00001 /**
00002  * @file framebuffer.h
00003  * @brief Буфер кадру, що використовується для рендеренгу.
00004  */
00005 #ifndef FRAMEBUFFER_H
00006 #define FRAMEBUFFER_H
00007
00008 #include "osdo.h"
00009 #include "glbindable.h"
00010
00011 class Renderbuffer;
00012 class Texture;
00013
00014 /**
00015  * @brief Буфер кадру, що використовується для рендеренгу.
00016  */
00017 class Framebuffer : public GlBindable
00018 {
00019     /**
00020      * @brief Віртуальний метод що створює буфер кадру.
00021      * @return індекс буфера кадру
00022      */
00023     GLuint _generate() const override;
00024     /**
00025      * @brief Віртуальний метод, що прив'язує контекст до буферу кадру.
00026      * @param id індекс буфера кадру
00027      * @param target ціль буфера кадру
00028      */
00029     virtual void _bind(const GLuint id, GLenum target) const override;
00030     /**
00031      * @brief Віртуальний метод, що задає ціль прив'язки за замовчуванням.
00032      * @return тип ціль прив'язки за замовчуванням
00033      */
00034     virtual GLenum _default() const override;
00035 public:
00036     Framebuffer();
00037     ~Framebuffer() override;
00038
00039     /**
00040      * @brief Перевіряє, чи готовий буфер до рендеренгу.
00041      * @param target тип буфера
00042      * @return статус буфера
00043      */
00044     bool check(GLenum target = GL_FRAMEBUFFER);
00045
00046     /**
00047      * @brief Підготовлює текстуру з згладжуванням
00048      * @param size ширина та висота кадру
00049      * @param texture текстура у яку зберігають кадр з згладжуванням
00050      */
00051     void tex_2d_multisample(GLsizei size[2], const Texture& texture);
00052
00053     /**
00054      * @brief Підготовлює звичайну текстуру
00055      * @param size ширина та висота кадру
00056      * @param texture текстура у яку зберігають кадр
00057      */
00058     void tex_2d(GLsizei size[2], const Texture& texture);
00059
00060     /**
00061      * @brief Підготовлює рендер буфер `rb`
00062      * @param rb рендер буфер
00063      * @param target тип буфера
00064      */
00065     void renderbuffer(const Renderbuffer& rb, GLenum target) const;
00066
00067     /**
00068      * @brief Підготовлює рендер буфер з згладжуванням для зберігання кольорів.
00069      * @param size ширина та висота кадру
00070      * @param rb рендер буфер
00071      */
00072     void rb_color_multisample(GLsizei size[2], const Renderbuffer& rb) const;
00073
00074     /**
00075      * @brief Підготовлює глибинний рендер буфер з згладжуванням.
00076      * @param size ширина та висота кадру
00077      * @param rb рендер буфер
00078      */
00079     void rb_depth_multisample(GLsizei size[2], const Renderbuffer& rb) const;
00080
00081     /**
00082      * @brief Підготовлює рендер буфер для зберігання кольорів.
00083      * @param size ширина та висота кадру
00084      * @param rb рендер буфер
00085      */
```

```
00086     void rb_color(GLsizei size[2], const Renderbuffer& rb) const;
00087
00088     /**
00089      * @brief Підготовлює глибинний рендер буфер.
00090      * @param size ширина та висота кадру
00091      * @param rb рендер буфер
00092      */
00093     void rb_depth(GLsizei size[2], const Renderbuffer& rb) const;
00094 };
00095
00096 #endif // FRAMEBUFFER_H
```

8.28 Файл osdo/glbindable.cpp

#include "glbindable.h"
#include "glbinder.h"
Діаграма включених заголовочних файлів для glbindable.cpp:



8.29 glbindable.cpp

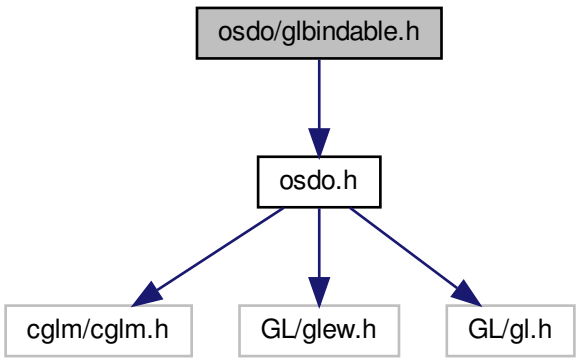
```
00001 #include "glbindable.h"
00002 #include "glbinder.h"
00003
00004 GLuint GlBindable::_generate() const {
00005     return 0;
00006 }
00007
00008 void GlBindable::_bind(const GLuint, GLenum) const {}
00009
00010 GLenum GlBindable::_default() const {
00011     return 0;
00012 }
00013
00014 GlBindable::GlBindable() : id(_generate()) {}
00015
00016 GlBindable::GlBindable(const GLuint id) : id(id) {}
00017
00018 GlBindable::~GlBindable() {}
00019
00020 const GLuint &GlBindable::get_id() const
00021 {
00022     return id;
```

```
00023 }
00024
00025 void *GlBindable::get_vid() const
00026 {
00027     return reinterpret_cast<void*>(static_cast<intptr_t>(id));
00028 }
00029
00030 void GlBindable::bind() const
00031 {
00032     _bind(id, _default());
00033 }
00034
00035 void GlBindable::bind(GLenum target) const
00036 {
00037     _bind(id, target);
00038 }
00039
00040 void GlBindable::unbind() const
00041 {
00042     _bind(0, _default());
00043 }
00044
00045 void GlBindable::unbind(GLenum target) const
00046 {
00047     _bind(0, target);
00048 }
00049
00050 GlBinder GlBindable::binder() const
00051 {
00052     return {*this, _default()};
00053 }
00054
00055 GlBinder GlBindable::binder(GLenum target) const
00056 {
00057     return {*this, target};
00058 }
00059
00060
```

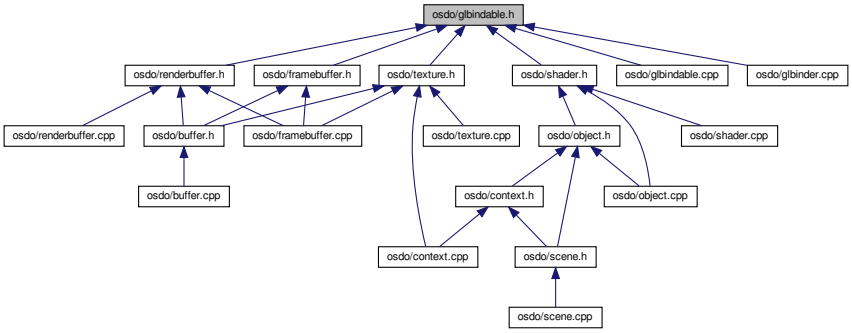
8.30 Файл osdo/glbindable.h

Абстрактний клас, який виконує роль генерації та прив'язки об'єктів OpenGL.

#include "osdo.h"
Діаграма включених заголовочних файлів для glbindable.h:



Граф файлів, які включають цей файл:



Класи

- class GlBindable

  Абстрактний клас, який виконує роль генерації та прив'язки об'єктів OpenGL.

8.30.1 Детальний опис

Абстрактний клас, який виконує роль генерації та прив'язки об'єктів OpenGL.

Див. визначення в файлі glbindable.h

8.31 glbindable.h

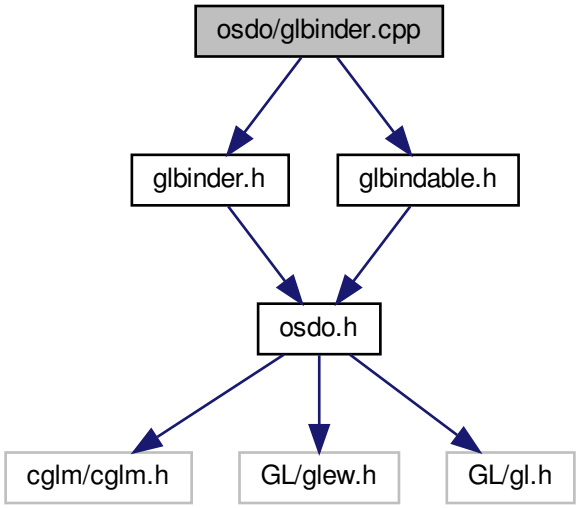
```
00001 /**
00002  * @file glbindable.h
00003  * @brief Абстрактний клас, який виконує роль генерації та прив’язки об’єктів OpenGL.
00004  */
00005 #ifndef GLBINDABLE_H
00006 #define GLBINDABLE_H
00007
00008 #include "osdo.h"
00009
00010 class GlBinder;
00011
00012 /**
00013  * @brief Абстрактний клас, який виконує роль генерації та прив’язки об’єктів OpenGL.
00014  */
00015 class GlBindable
00016 {
00017     /**
00018      * @brief Індекс об’єкту OpenGL.
00019      */
00020     const GLuint id;
00021 protected:
00022     /**
00023      * @brief Віртуальний метод що створює об’єкт.
00024      * @return індекс об’єкту OpenGL
00025      */
00026     virtual GLuint _generate() const;
00027     /**
00028      * @brief Віртуальний метод, що прив’язує контекст OpenGL до об’єкту.
00029      * @param id індекс об’єкту OpenGL
00030      * @param target ціль прив’язки об’єкту
00031      */
00032     virtual void _bind(const GLuint id, GLenum target) const;
00033     /**
00034      * @brief Віртуальний метод, що задає ціль прив’язки за замовчуванням.
00035      * @return ціль прив’язки за замовчуванням
00036      */
00037     virtual GLenum _default() const;
```

```
00038 protected:
00039     GlBindable();
00040     /**
00041      * @brief Конструктор з параметром.
00042      * @param id індекс об'єкту OpenGL
00043      */
00044     GlBindable(const GLuint id);
00045 public:
00046     virtual ~GlBindable();
00047
00048     GlBindable(const GlBindable&) = delete;
00049     GlBindable(GlBindable&&) = delete;
00050     GlBindable& operator=(const GlBindable&) = delete;
00051     GlBindable& operator=(GlBindable&&) = delete;
00052
00053     /**
00054      * @brief Повертає індекс об'єкту OpenGL.
00055      * @return індекс об'єкту OpenGL
00056      */
00057     const GLuint& get_id() const;
00058     /**
00059      * @brief Повертає індекс об'єкту OpenGL у типі 'void*'
00060      * @return індекс об'єкту OpenGL
00061      */
00062     void* get_vid() const;
00063
00064     /**
00065      * @brief Прив'язує контекст до об'єкту із ціллю за замовчуванням.
00066      */
00067     void bind() const;
00068     /**
00069      * @brief Прив'язує контекст до об'єкту із ціллю 'target'
00070      * @param target ціль прив'язки об'єкту
00071      */
00072     void bind(GLenum target) const;
00073     /**
00074      * @brief Відв'язує контекст до об'єкту із ціллю за замовчуванням.
00075      */
00076     void unbind() const;
00077     /**
00078      * @brief Відв'язує контекст до об'єкту із ціллю 'target'
00079      * @param target ціль прив'язки об'єкту
00080      */
00081     void unbind(GLenum target) const;
00082
00083     /**
00084      * @brief Повертає об'єкт прив'язки контексту із ціллю за замовчуванням
00085      * @return об'єкт прив'язки контексту із ціллю за замовчуванням
00086      */
00087     GlBinder binder() const;
00088     /**
00089      * @brief Повертає об'єкт прив'язки контексту із ціллю 'target'
00090      * @param target ціль прив'язки об'єкту
00091      * @return об'єкт прив'язки контексту із ціллю 'target'
00092      */
00093     GlBinder binder(GLenum target) const;
00094 };
00095
00096 #endif // GLBINDABLE_H
```

8.32 Файл osdo/glbinder.cpp

```
#include "glbinder.h"
#include "glbindable.h"
```

Діаграма включених заголовочних файлів для glbinder.cpp:



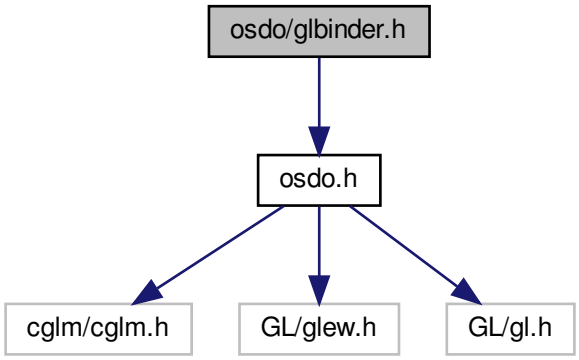
8.33 glbinder.cpp

```
00001 #include "glbinder.h"
00002 #include "glbindable.h"
00003
00004
00005 GlBinder::GlBinder(const GlBindable &bindable, GLenum target)
00006     : bindable(bindable), target(target) {
00007     bindable.bind(target);
00008 }
00009
00010 GlBinder::~GlBinder() {bindable.unbind(target);}
```

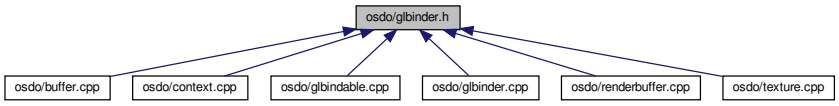
8.34 Файл osdo/glbinder.h

Клас який прив'язує контексту до деякого об'єкту OpenGL.

#include "osdo.h"
Діаграма включених заголовочних файлів для glbinder.h:



Граф файлів, які включають цей файл:



Класи

- class GlBinder

  Клас який прив'язує контексту до деякого об'єкту OpenGL.

8.34.1 Детальний опис

Клас який прив'язує контексту до деякого об'єкту OpenGL.

Див. визначення в файлі glbinder.h

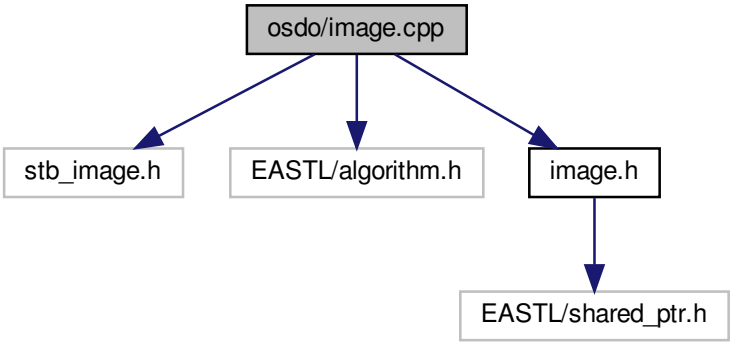
8.35 glbinder.h

```
00001 /**
00002  * @file glbinder.h
00003  * @brief Клас який прив'язує контексту до деякого об'єкту OpenGL.
00004  */
00005 #ifndef GLBINDER_H
00006 #define GLBINDER_H
00007
00008 #include "osdo.h"
00009
00010 class GlBindable;
00011
00012 /**
```

```
00013  * @brief Клас який прив'язує контексту до деякого об'єкту OpenGL.
00014  */
00015 class GlBinder
00016 {
00017     /**
00018      * @brief Об'єкт OpenGL, який прив'язують.
00019      */
00020     const GlBindable& bindable;
00021     /**
00022      * @brief Ціль прив'язки.
00023      */
00024     const GLenum target;
00025 public:
00026     /**
00027      * @brief Конструктор із параметром
00028      * @param bindable об'єкт OpenGL, який прив'язують
00029      * @param target ціль прив'язки
00030      */
00031     GlBinder(const GlBindable& bindable, GLenum target);
00032     ~GlBinder();
00033 };
00034
00035 #endif // GLBINDER_H
```

8.36 Файл osdo/image.cpp

#include <stb_image.h>
#include <EASTL/algorithm.h>
#include "image.h"
Діаграма включених заголовочних файлів для image.cpp:



Макровизначення

- #define STB_IMAGE_IMPLEMENTATION

8.36.1 Опис макровизначень

8.36.1.1 STB_IMAGE_IMPLEMENTATION #define STB_IMAGE_IMPLEMENTATION

Див. визначення в файлі image.cpp, рядок 1

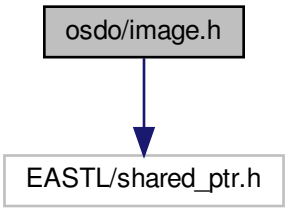
8.37 image.cpp

```
00001 #define STB_IMAGE_IMPLEMENTATION
00002 #include <stb_image.h>
00003 #include <EASTL/algorithm.h>
00004 #include "image.h"
00005
00006 using eastl::min;
00007 using eastl::max;
00008
00009 Image::Image(shared_ptr<const pixel_t[]> data,
00010              const int width, const int height)
00011     : data(data), width(width), height(height)
00012 {
00013 }
00014
00015 Image Image::fromFile(const char *path)
00016 {
00017     int width = 0, height = 0;
00018     pixel_t (*data)[] = (pixel_t (*)[])stbi_load(path, &width, &height, nullptr, COMP);
00019     if (data) {
00020         shared_ptr<pixel_t[]> ptr(data, stbi_image_free);
00021         return {ptr, width, height};
00022     }
00023     return {nullptr, 0, 0};
00024 }
```

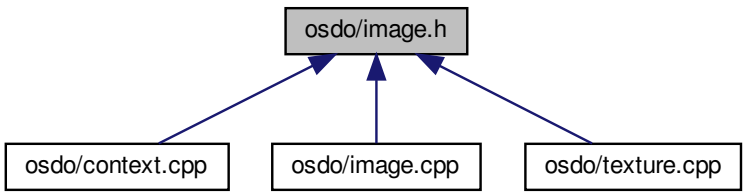
8.38 Файл osdo/image.h

Задає клас, який зберігає масив пікселів, ширину та висоту.

#include <EASTL/shared_ptr.h>
Діаграма включених заголовочних файлів для image.h:



Граф файлів, які включають цей файл:



Класи

- class Image

  Зберігає масив пікселів, ширину та висоту.

Макровизначення

- #define COMP 4

Визначення типів

- typedef unsigned char channel_t

  Тип каналу зображення. Використовується unsigned char, оскільки значення нашого каналу знаходяться в діапазоні [0, 255], таким чином канал займає 8-біт.
- typedef channel_t pixel_t[COMP]

  Тип пікселя. Один піксель має COMP каналів і канал визначений типом channel_t. Використовується 8-бітний канал зі значеннями, що знаходяться в діапазоні [0, 255]. Піксель має 4 канали (червоний, зелений, синій та альфа), тому займає 4 байти.

8.38.1 Детальний опис

Задає клас, який зберігає масив пікселів, ширину та висоту.

Див. визначення в файлі image.h

8.38.2 Опис макровизначень

8.38.2.1 COMP #define COMP 4

Розмір пікселя у байтах

Назва означає "компоненти" та взято з бібліотеки файлів заголовка stb_image.h.

Див. визначення в файлі image.h, рядок 14

8.38.3 Опис визначень типів

8.38.3.1 channel_t typedef unsigned char channel_t

Тип каналу зображення. Використовується unsigned char, оскільки значення нашого каналу знаходяться в діапазоні [0, 255], таким чином канал займає 8-біт.

Див. визначення в файлі image.h, рядок 23

8.38.3.2 pixel_t typedef channel_t pixel_t[COMP]

Тип пікселя. Один піксель має COMP каналів і канал визначений типом channel_t. Використовується 8-бітний канал зі значеннями, що знаходяться в діапазоні [0, 255]. Піксель має 4 канали (червоний, зелений, синій та альфа), тому займає 4 байти.

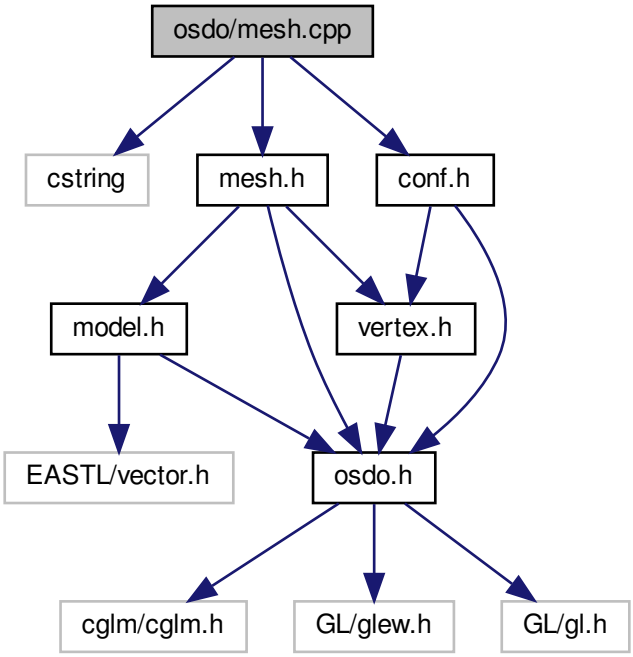
Див. визначення в файлі image.h, рядок 31

8.39 image.h

```
00001 /**
00002  * @file image.h
00003  * @brief Задає клас, який зберігає масив пікселів, ширину та висоту.
00004  */
00005 #ifndef IMAGE_H
00006 #define IMAGE_H
00007 #include <EASTL/shared_ptr.h>
00008
00009 /**
00010  * Розмір пікселя у байтах
00011  *
00012  * Назва означає "компоненти" та взято з бібліотеки файлів заголовка ‘stb_image.h‘.
00013  */
00014 #define COMP 4
00015
00016 using eastl::shared_ptr;
00017
00018 /**
00019  * @brief Тип каналу зображення.
00020  * Використовується ‘unsigned char‘, оскільки значення нашого каналу
00021  * знаходяться в діапазоні [0, 255], таким чином канал займає 8-біт.
00022  */
00023 typedef unsigned char channel_t;
00024
00025 /**
00026  * @brief Тип пікселя.
00027  * Один піксель має ‘COMP‘ каналів і канал визначений типом ‘channel_t‘.
00028  * Використовується 8-бітний канал зі значеннями, що знаходяться в діапазоні [0, 255].
00029  * Піксель має 4 канали (червоний, зелений, синій та альфа), тому займає 4 байти.
00030  */
00031 typedef channel_t pixel_t[COMP];
00032
00033 /**
00034  * @brief Зберігає масив пікселів, ширину та висоту.
00035  */
00036 class Image
00037 {
00038 public:
00039     /**
00040      * @brief Константний масив пікселів.
00041      * Зберігає розумний вказівник на масив пікселів зображення
00042      * розміром висота*ширина.
00043      */
00044     shared_ptr<const pixel_t[]> data;
00045     /**
00046      * @brief Ширина зображення.
00047      */
00048     const int width;
00049     /**
00050      * @brief Висота зображення.
00051      */
00052     const int height;
00053
00054     /**
00055      * @brief Конструктор для зображення.
00056      * Зберігає посилання на масив пікселів,
00057      * а також висоту та ширину зображення.
00058      * @param data Розумне посилання на масив пікселів,
00059      * розмір повинен бути ‘висота * ширина‘.
00060      * @param width Ширина зображення.
00061      * @param height Висота зображення.
00062      */
00063     Image(shared_ptr<const pixel_t[]> data,
00064           const int width, const int height);
00065     /**
00066      * @brief Зчитування зображення з файлу.
00067      * @param path Шлях до файлу зображення.
00068      * @return клас ‘Image‘, який посилається на пікселі.
00069      */
00070     static Image fromFile(const char *path);
00071 };
00072
00073
00074 #endif // IMAGE_H
```

8.40 Файл osdo/mesh.cpp

#include <cstring>
#include "mesh.h"
#include "conf.h"
Діаграма включених заголовочних файлів для mesh.cpp:



8.41 mesh.cpp

```
00001 #include <cstring>
00002 #include "mesh.h"
00003 #include "conf.h"
00004
00005 Mesh::Mesh() : indices_size(0) {
00006     // create buffers/arrays
00007     glGenVertexArrays(1, &this->vao);
00008     glGenBuffers(1, &this->vbo);
00009     glGenBuffers(1, &this->ebo);
00010     glBindVertexArray(this->vao);
00011     glBindBuffer(GL_ARRAY_BUFFER, this->vbo);
00012     glBindBuffer(GL_ELEMENT_ARRAY_BUFFER, this->ebo);
00013
00014     // set the vertex attribute pointers
00015     // vertex Positions
00016     glEnableVertexAttribArray(0);
00017     glVertexAttribPointer(0, 3, GL_FLOAT, GL_FALSE, sizeof(Vertex),
00018                           reinterpret_cast<void*>(offsetof(Vertex, position)));
00019
00020     // vertex normals
00021     glEnableVertexAttribArray(1);
00022     glVertexAttribPointer(1, 3, GL_FLOAT, GL_FALSE, sizeof(Vertex),
00023                           reinterpret_cast<void*>(offsetof(Vertex, normal)));
00024
00025     // vertex color
00026     glEnableVertexAttribArray(2);
00027     glVertexAttribPointer(2, 4, GL_UNSIGNED_BYTE, GL_TRUE, sizeof(Vertex),
00028                           reinterpret_cast<void*>(offsetof(Vertex, color)));
00029
```

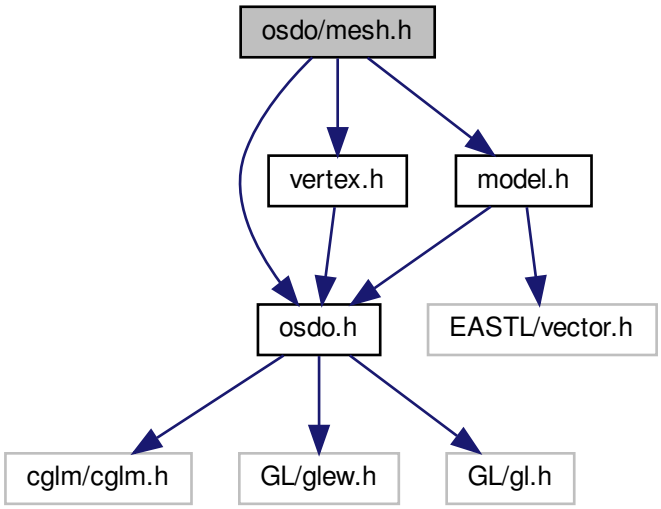
```
00030     // vertex uv
00031     glEnableVertexAttribArray(3);
00032     glVertexAttribPointer(3, 2, GL_FLOAT, GL_FALSE, sizeof(Vertex),
00033                           reinterpret_cast<void*>(offsetof(Vertex, uv)));
00034
00035     glBindBuffer(GL_ARRAY_BUFFER, 0);
00036     glBindBuffer(GL_ELEMENT_ARRAY_BUFFER, 0);
00037     glBindVertexArray(0);
00038 }
00039
00040 Mesh::~Mesh() {
00041     glDeleteVertexArrays(1, &this->vao);
00042     glDeleteBuffers(1, &this->vbo);
00043     glDeleteBuffers(1, &this->ebo);
00044 }
00045
00046 void Mesh::cube_update() {
00047     size_t vert_size = sizeof(EXAMPLE_CUBE_VERTEX) / sizeof(Vertex);
00048     size_t indi_size = sizeof(EXAMPLE_CUBE_INDICIES) / sizeof(GLuint);
00049     update(EXAMPLE_CUBE_VERTEX, vert_size,
00050            EXAMPLE_CUBE_INDICIES, indi_size);
00051 }
00052
00053 void Mesh::update(const Vertex *vertices, size_t vertices_n,
00054                   const GLuint *indices, size_t indices_n) {
00055     if (vertices_n == 0 || indices_n == 0)
00056         return;
00057     indices_size = static_cast<GLint>(indices_n);
00058     glBindVertexArray(this->vao);
00059     // load data into vertex buffers
00060     glBindBuffer(GL_ARRAY_BUFFER, this->vbo);
00061     glBufferData(GL_ARRAY_BUFFER, vertices_n * sizeof(Vertex),
00062                  vertices, GL_STATIC_DRAW);
00063
00064     glBindBuffer(GL_ELEMENT_ARRAY_BUFFER, this->ebo);
00065     glBufferData(GL_ELEMENT_ARRAY_BUFFER, indices_n * sizeof(GLuint),
00066                  indices, GL_STATIC_DRAW);
00067
00068     glBindBuffer(GL_ARRAY_BUFFER, 0);
00069     glBindBuffer(GL_ELEMENT_ARRAY_BUFFER, 0);
00070     glBindVertexArray(0);
00071 }
00072
00073 void Mesh::draw_mode(GLenum mode) {
00074     if (!indices_size)
00075         return;
00076     glBindVertexArray(this->vao);
00077     glBindBuffer(GL_ELEMENT_ARRAY_BUFFER, this->ebo);
00078     glBindBuffer(GL_ARRAY_BUFFER, this->vbo);
00079     glDrawElements(mode, static_cast<GLint>(this->indices_size),
00080                    GL_UNSIGNED_INT, nullptr);
00081
00082     glBindBuffer(GL_ARRAY_BUFFER, 0);
00083     glBindBuffer(GL_ELEMENT_ARRAY_BUFFER, 0);
00084     glBindVertexArray(0);
00085 }
00086
00087 void Mesh::draw(Shader &, bool) {
00088     Mesh::draw_mode(GL_TRIANGLES);
00089 }
```

8.42 Файл osdo/mesh.h

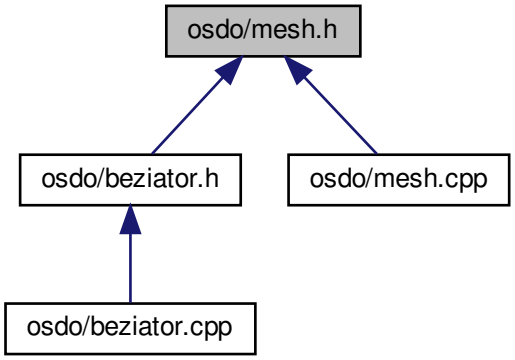
Задає клас мешу, який зберігається на відеокарті.

```
#include "osdo.h"
#include "model.h"
#include "vertex.h"
```

Діаграма включених заголовочних файлів для mesh.h:



Граф файлів, які включають цей файл:



Класи

- class Mesh

  Меш, який зберігається на відеокарті.

8.42.1 Детальний опис

Задає клас мешу, який зберігається на відеокарті.

Див. визначення в файлі mesh.h

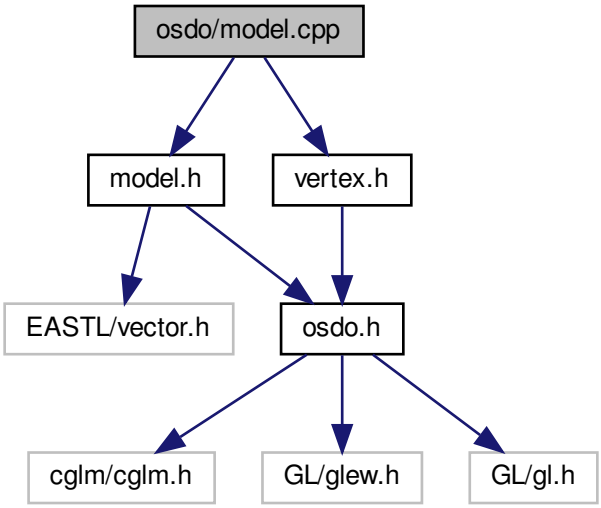
8.43 mesh.h

```
00001 /**
00002  * @file mesh.h
00003  * @brief Задає клас мешу, який зберігається на відеокарті.
00004  */
00005 #ifndef MESH_H
00006 #define MESH_H
00007
00008 #include "osdo.h"
00009 #include "model.h"
00010 #include "vertex.h"
00011
00012 /**
00013  * @brief Меш, який зберігається на відеокарті.
00014  */
00015 class Mesh : public Model {
00016 protected:
00017     /**
00018      * @brief Об'єкт масиву вершин. Розшифровується "Vertex Array Object".
00019      */
00020     GLuint vao;
00021     /**
00022      * @brief Об'єкт буфера вершин. Розшифровується "Vertex Buffer Object".
00023      */
00024     GLuint vbo;
00025     /**
00026      * @brief Об'єкти буфера елементів. Розшифровується "Element Buffer Objects".
00027      */
00028     GLuint ebo;
00029     /**
00030      * @brief Кількість індексів у ‘ebo‘.
00031      */
00032     GLint indices_size;
00033 public:
00034     Mesh();
00035     ~Mesh() override;
00036
00037     Mesh(const Mesh&) = delete;
00038     Mesh(Mesh&&) = delete;
00039     Mesh& operator=(const Mesh&) = delete;
00040     Mesh& operator=(Mesh&&) = delete;
00041
00042     /**
00043      * @brief Завантажити у відеокарту меш куба.
00044      */
00045     void cube_update();
00046     /**
00047      * @brief Завантажити у відеокарту новий меш.
00048      * @param vertices масив вершин
00049      * @param vertices_n кількість вершин
00050      * @param indices масив індексів вершин
00051      * @param indices_n кількість індексів вершин
00052      */
00053     void update(const Vertex* vertices, size_t vertices_n,
00054                 const GLuint* indices, size_t indices_n);
00055
00056     /**
00057      * @brief Відображує меш.
00058      * @param shader Шейдер який використовуєтсья для відображення мешу.
00059      * @param pre_generated флаг залишений для інтерфесу, але не використовується.
00060      */
00061     void draw(Shader &shader, bool pre_generated) override;
00062     /**
00063      * @brief Відображує меш у певному режимі.
00064      * Див. [glDrawElements][glDrawElements].
00065      * [glDrawElements]: https://www.khronos.org/registry/OpenGL-
      Refpages/gl4/html/glDrawElements.xhtml#parameters
00066      * @param mode режим відображення.
00067      */
00068     void draw_mode(GLenum mode);
00069 };
00070
00071 #endif
```

8.44 Файл osdo/model.cpp

```
#include "model.h"
#include "vertex.h"
```

Діаграма включених заголовочних файлів для model.cpp:



8.45 model.cpp

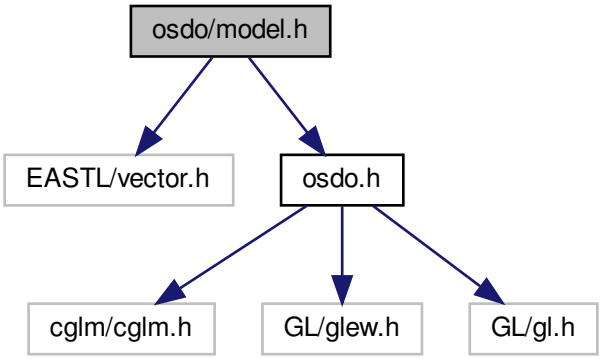
```
00001 #include "model.h"
00002 #include "vertex.h"
00003
00004 Model::~Model() {}
00005
00006 void Model::draw(Shader &, bool pre_generated) {}
00007
00008 void Model::generate(size_t d) {}
00009
00010 vector<Vertex> *Model::get_vertices() {
00011     return nullptr;
00012 }
00013
00014 void Model::edit_panel() {}
```

8.46 Файл osdo/model.h

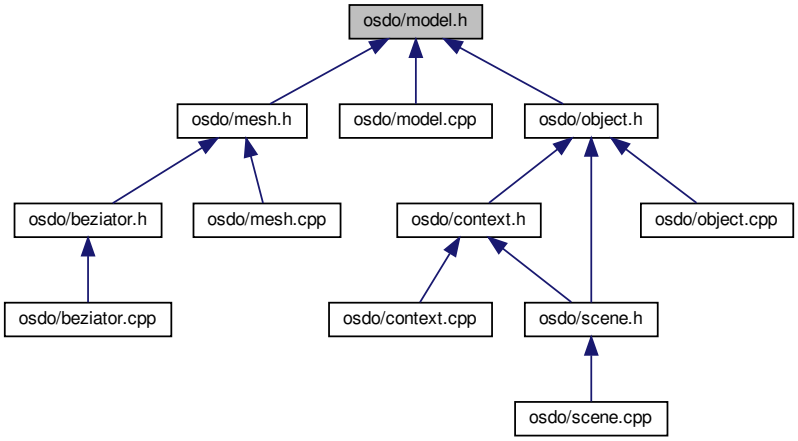
Задає інтерфейс моделі, яку можна відобразити.

#include <EASTL/vector.h>
#include "osdo.h"

Діаграма включених заголовочних файлів для model.h:



Граф файлів, які включають цей файл:



Класи

- class Model

  Інтерфейс до деякої моделі, яку можна відобразити.

8.46.1 Детальний опис

Задає інтерфейс моделі, яку можна відобразити.

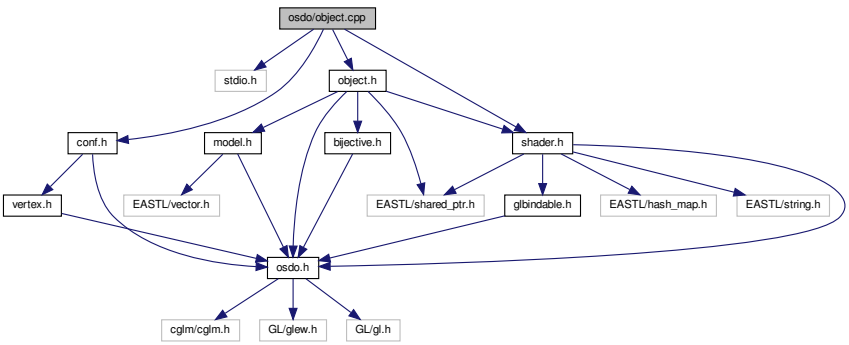
Див. визначення в файлі model.h

8.47 model.h

```
00001 /**
00002  * @file model.h
00003  * @brief Задає інтерфейс моделі, яку можна відобразити.
00004  */
00005 #ifndef MODEL_H
00006 #define MODEL_H
00007
00008 #include <EASTL/vector.h>
00009 #include "osdo.h"
00010
00011 struct Vertex;
00012 class Shader;
00013 using eastl::vector;
00014
00015 /**
00016  * @brief Інтерфейс до деякої моделі, яку можна відобразити.
00017  */
00018 class Model {
00019 public:
00020     virtual ~Model();
00021     /**
00022      * @brief Відображує модель.
00023      * @param shader Шейдер який використовуєтсья для відображення моделі.
00024      * @param pre_generated флаг, який позначає яким чином відображати модель.
00025      */
00026     virtual void draw(Shader &shader, bool pre_generated = false);
00027     /**
00028      * @brief Генерує деталізований меш моделі.
00029      * Див. ‘Beziator::generate‘
00030      * @param d ступінь деталізації.
00031      */
00032     virtual void generate(size_t d = 8);
00033     /**
00034      * @brief Видає список вершин моделі.
00035      * @return Вказівник на поле ‘vertices‘.
00036      */
00037     virtual vector<Vertex> *get_vertices();
00038     /**
00039      * @brief Створює вікно редагування моделі.
00040      */
00041     virtual void edit_panel();
00042 };
00043
00044 #endif // MODEL_H
```

8.48 Файл osdo/object.cpp

#include <stdio.h>
#include "object.h"
#include "conf.h"
#include "shader.h"
Діаграма включених заголовочних файлів для object.cpp:



8.49 object.cpp

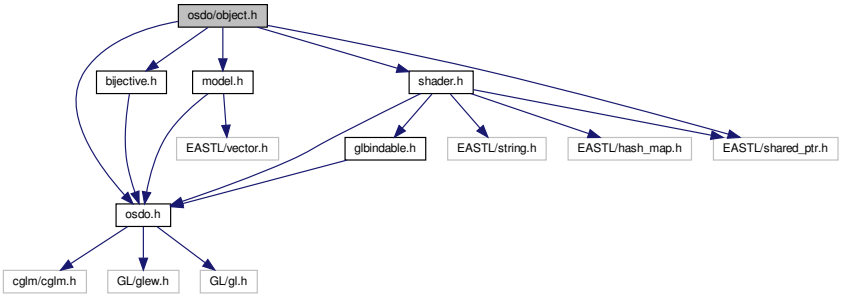
```
00001 #include <stdio.h>
00002
00003 #include "object.h"
00004 #include "conf.h"
00005 #include "shader.h"
00006
00007 Object::Object(shared_ptr<Model> model)
00008     : transform GLM_MAT4_IDENTITY_INIT,
00009       position GLM_VEC4_BLACK_INIT,
00010       animation GLM_VEC3_ZERO_INIT,
00011       model(model)
00012 {
00013 }
00014 void Object::get_position(vec4 position)
00015 {
00016     glm_vec4_copy(this->position, position);
00017 }
00018
00019 void Object::set_position(vec4 position)
00020 {
00021     glm_vec4_copy(position, this->position);
00022 }
00023
00024 void Object::get_rotation(vec3 rotation)
00025 {
00026     glm_euler_angles(this->transform, rotation);
00027 }
00028
00029 void Object::set_rotation(vec3 rotation)
00030 {
00031     glm_euler_xyz(rotation, this->transform);
00032 }
00033
00034 void Object::get_animation(vec3 animation)
00035 {
00036     glm_vec3_copy(this->animation, animation);
00037 }
00038
00039 void Object::set_animation(vec3 animation)
00040 {
00041     glm_vec3_copy(animation, this->animation);
00042 }
00043
00044 void Object::get_mat4(mat4 dest) {
00045     glm_translate_make(dest, this->position);
00046     glm_mat4_mul(dest, this->transform, dest);
00047 }
00048
00049 void Object::translate(vec3 distances, float delta_time) {
00050     vec3 new_distances = GLM_VEC3_ZERO_INIT;
00051     glm_vec3_muladds(distances, OBJECT_MOVE_SPEED * delta_time,
00052                      new_distances);
00053     Object::translate_object( new_distances);
00054 }
00055
00056 void Object::draw(Shader &shader, mat4 mat4buf, GLdouble delta_time,
00057                   bool pre_generated) {
00058     // render the loaded model
00059     Object::animate(static_cast<GLfloat>(delta_time));
00060     Object::get_mat4(mat4buf);
00061     shader.set_mat4("model", mat4buf);
00062     this->model->draw(shader, pre_generated);
00063 }
00064
00065 void Object::rotate(enum coord_enum coord, float delta_time) {
00066     Object::rotate_object(delta_time * OBJECT_ROTATE_SPEED, coord);
00067 }
00068
00069 void Object::rotate_all(vec3 angles) {
00070     Object::rotate_all_object(angles);
00071 }
00072
00073 void Object::add_animation(vec3 angles, float delta_time) {
00074     vec3 animation = GLM_VEC3_ZERO_INIT;
00075     glm_vec3_muladds(angles, delta_time, animation);
00076     glm_vec3_add(this->animation, animation,
00077                  this->animation);
00078 }
00079
00080 shared_ptr<Model> Object::get_model()
00081 {
00082     return model;
00083 }
00084
00085 void Object::translate_object(vec3 distances) {
```

```
00086     glm_vec3_add(this->position, distances, this->position);
00087 }
00088
00089 void Object::rotate_object(float angle, enum coord_enum coord) {
00090     mat4 matrix = GLM_MAT4_IDENTITY_INIT;
00091     switch (coord) {
00092     case X: glm_rotate_x(matrix, angle, matrix); break;
00093     case Y: glm_rotate_y(matrix, angle, matrix); break;
00094     case Z: glm_rotate_z(matrix, angle, matrix); break;
00095     }
00096     glm_mat4_mul(matrix, this->transform, this->transform);
00097 }
00098
00099 void Object::rotate_all_object(vec3 angles) {
00100     Object::rotate_object(angles[0], X);
00101     Object::rotate_object(angles[1], Y);
00102     Object::rotate_object(angles[2], Z);
00103 }
00104
00105 void Object::animate(float step) {
00106     vec3 animation = GLM_VEC3_ZERO_INIT;
00107     glm_vec3_muladds(this->animation, step, animation);
00108     Object::rotate_all(animation);
00109 }
00110
00111 void Object::scale(vec3 scale) {
00112     glm_scale(this->transform, scale);
00113 }
00114
00115 mat4 *Object::get_transform()
00116 {
00117     return &this->transform;
00118 }
```

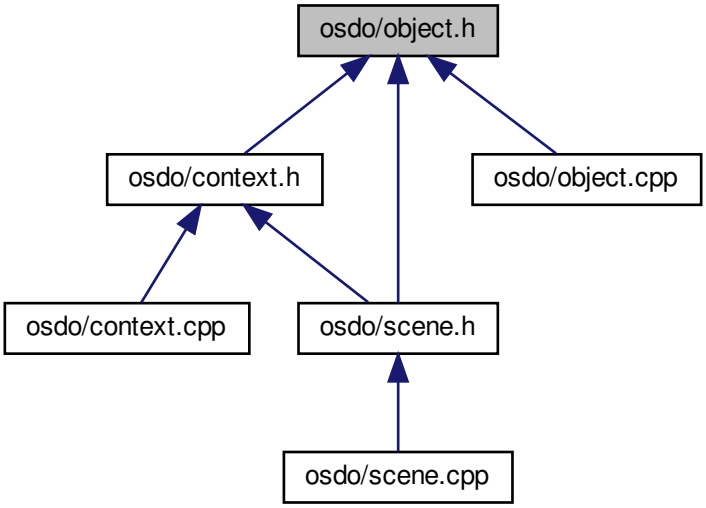
8.50 Файл osdo/object.h

Задає клас об'єкту моделі.

#include "osdo.h"
#include "bijective.h"
#include "model.h"
#include "shader.h"
#include "EASTL/shared_ptr.h"
Діаграма включених заголовочних файлів для object.h:



Граф файлів, які включають цей файл:



Класи

- class Object

  Об'єкт моделі.

8.50.1 Детальний опис

Задає клас об'єкту моделі.

Див. визначення в файлі object.h

8.51 object.h

```
00001 /**
00002  * @file object.h
00003  * @brief Задає клас об'єкту моделі.
00004  */
00005 #ifndef OBJECT_H
00006 #define OBJECT_H
00007
00008 #include "osdo.h"
00009
00010 #include "bijective.h"
00011 #include "model.h"
00012 #include "shader.h"
00013 #include "EASTL/shared_ptr.h"
00014 using eastl::shared_ptr;
00015 using eastl::make_shared;
00016
00017 /**
00018  * @brief Об'єкт моделі.
00019  */
00020 class Object : public Bijective {
00021     /**
00022      * @brief Матриця лінійних перетворень.
00023      */
```

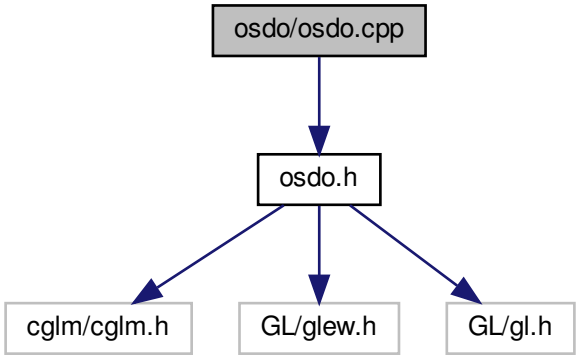
```
00024     mat4 transform;
00025     /**
00026      * @brief Позиція об'єкту у просторі
00027      */
00028     vec4 position;
00029     /**
00030      * @brief Анімація обертання по осям \f$(x, y, z, 1.0)\f$.
00031      */
00032     vec4 animation;
00033     /**
00034      * @brief Модель об'єкту.
00035      */
00036     shared_ptr<Model> model;
00037
00038 public:
00039     /**
00040      * @brief Конструктор, який створює об'єкт з моделі.
00041      * @param model Модель об'єкту.
00042      */
00043     Object(shared_ptr<Model> model = nullptr);
00044     ~Object() override = default;
00045
00046     /**
00047      * @brief Забирає поточну позицію об'єкта у просторі.
00048      * @param[out] position поточна позицію об'єкта
00049      */
00050     void get_position(vec4 position) override;
00051     /**
00052      * @brief Задає нову позицію об'єкта у просторі.
00053      * @param[in] position нова позиція об'єкта у просторі
00054      */
00055     void set_position(vec4 position) override;
00056
00057     /**
00058      * @brief Забирає поточний нахил об'єкта.
00059      * @param[out] rotation поточний нахил  об'єкта
00060      */
00061     void get_rotation(vec3 rotation) override;
00062     /**
00063      * @brief Задає новий нахил об'єкта.
00064      * @param[in] rotation новий нахил об'єкта
00065      */
00066     void set_rotation(vec3 rotation) override;
00067
00068     /**
00069      * @brief Забирає поточну анімацію обернення об'єкта.
00070      * @param[out] rotation поточна анімація обернення об'єкта
00071      */
00072     void get_animation(vec3 rotation) override;
00073     /**
00074      * @brief Задає нову анімацію обернення об'єкта.
00075      * @param[in] rotation нова анімація обернення об'єкта.
00076      */
00077     void set_animation(vec3 rotation) override;
00078
00079     /**
00080      * @brief Забирає матрицю лінійних перетворень над об'єктом.
00081      * @param[out] matrix матриця лінійних перетворень
00082      */
00083     void get_mat4(mat4 matrix) override;
00084
00085     /**
00086      * @brief Переміщує об'єкт у просторі.
00087      *
00088      * Переміщує об'єкт у просторі на відстані з аргументу 'distances',
00089      * де кожне значення вектору позначає відстань відповідної осі.
00090      * @param[in] distances відстані переміщення по осям
00091      * @param[in] delta_time скільки часу пройшло з останнього кадру
00092      */
00093     void translate(vec3 distances, float delta_time) override;
00094     /**
00095      * @brief Обертає об'єкт.
00096      * @param[in] coord позначає координатну вісь навколо якої обертати
00097      * @param[in] delta_time скільки часу пройшло з останнього кадру
00098      */
00099     void rotate(enum coord_enum coord, float delta_time) override;
00100     /**
00101      * @brief Обернути об'єкт по всім осям.
00102      * @param[in] angles вектор кутів у радіанах на кожну вісь
00103      */
00104     void rotate_all(vec3 angles) override;
00105     /**
00106      * @brief Додає швидкість анімації обертання об'єкту.
00107      * @param[in] angles вектор швидкостей анімацій обертання по трьом осям
00108      * @param[in] delta_time скільки часу пройшло з останнього кадру
00109      */
00110     void add_animation(vec3 angles, float delta_time) override;
```

```
00111
00112     /**
00113      * @brief Повертає модель об'єкту.
00114      * @return модель об'єкту
00115      */
00116     shared_ptr<Model> get_model();
00117
00118     /**
00119      * @brief Відображує об'єкт
00120      * @param shader Шейдер який використовуєтсья для відображення моделі
00121      * @param mat4buf буфер матриці
00122      * @param delta_time скільки часу пройшло з останнього кадру
00123      * @param pre_generated флаг, який позначає яким чином відображати модель
00124      */
00125     void draw(Shader &shader, mat4 mat4buf, GLdouble delta_time,
00126               bool pre_generated);
00127
00128     /**
00129      * @brief Переміщує об'єкт незалежно від часу.
00130      * @param distances відстані переміщення по трьом осям \f$(x, y, z)\f$
00131      */
00132     void translate_object(vec3 distances);
00133
00134     /**
00135      * @brief Обертає об'єкт по осі незалежно від часу.
00136      * @param angle кут обертання у радіанах
00137      * @param coord ввісь обертання
00138      */
00139     void rotate_object(float angle, enum coord_enum coord);
00140     /**
00141      * @brief Обертає об'єкт по всім осям незалежно від часу.
00142      * @param angles вектор кутів обертання по трьом осям \f$(x, y, z)\f$
00143      */
00144     void rotate_all_object(vec3 angles);
00145
00146     /**
00147      * @brief Обертає об'єкт по заданій анімації.
00148      * @param step шаг анімації
00149      */
00150     void animate(float step);
00151
00152     /**
00153      * @brief Збільщує або зменшує об'єкт по трьом осям.
00154      * @param scale коеффіціенти зміни розміру по осям \f$(x, y, z)\f$
00155      */
00156     void scale(vec3 scale);
00157     /**
00158      * @brief Повертає матрицю лінійних перетворень без переміщення.
00159      * @return матриця лінійних перетворень
00160      */
00161     mat4* get_transform();
00162 };
00163
00164 #endif // OBJECT_H
```

8.52 Файл osdo/osdo.cpp

#include "osdo.h"

Діаграма включених заголовочних файлів для osdo.cpp:



Функції

- void ∗ operator new[] (size_t size, const char ∗name, int flags, unsigned debugFlags, const char ∗file, int line)

  Перевантажений оператор new для EASTL.
- void ∗ operator new[] (size_t size, size_t alignment, size_t alignmentOffset, const char ∗pName, int flags, unsigned debugFlags, const char ∗file, int line)

  Перевантажений оператор new для EASTL.

Змінні

- vec3 BASIS0POS = { 0.0f, 0.0f,-32.0f}

  Початкова позиція камери.

8.52.1 Опис функцій

8.52.1.1 operator new[]() [1/2]

```
void* operator new[] (
    size_t size,
    const char * name,
    int flags,
    unsigned debugFlags,
    const char * file,
    int line )
```

Перевантажений оператор new для EASTL.

Див. визначення в файлі osdo.cpp, рядок 3

8.52.1.2 operator new[]() [2/2] void* operator new[] (

size_t size,
size_t alignment,
size_t alignmentOffset,
const char * pName,
int flags,
unsigned debugFlags,
const char * file,
int line)

Перевантажений оператор new для EASTL.

Див. визначення в файлі osdo.cpp, рядок 8

8.52.2 Опис змінних

8.52.2.1 BASIS0POS vec3 BASIS0POS = { 0.0f, 0.0f,-32.0f}

Початкова позиція камери.

Див. визначення в файлі osdo.cpp, рядок 12

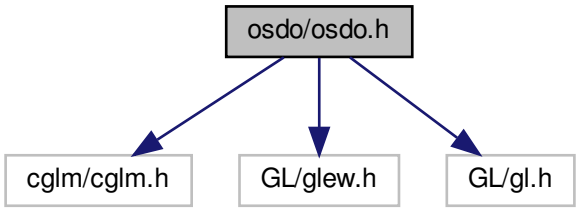
8.53 osdo.cpp

```
00001 #include "osdo.h"
00002
00003 void* operator new[](size_t size, const char* name, int flags, unsigned debugFlags, const char* file, int line)
00004 {
00005     return malloc(size);
00006 }
00007
00008 void* operator new[](size_t size, size_t alignment, size_t alignmentOffset, const char* pName, int flags, unsigned
        debugFlags, const char* file, int line) {
00009     return malloc(size);
00010 }
00011
00012 vec3 BASIS0POS = { 0.0f, 0.0f,-32.0f};
00013 //vec3 BASIS1POS = {-8.0f, 0.0f,  0.0f};
00014 //vec3 BASIS2POS = { 8.0f, 0.0f,  0.0f};
00015
00016 //vec3 BASIS1ROT = { 0.0f, 0.0f,  0.2f};
00017 //vec3 BASIS2ROT = { 0.0f, 0.0f, -0.2f};
```

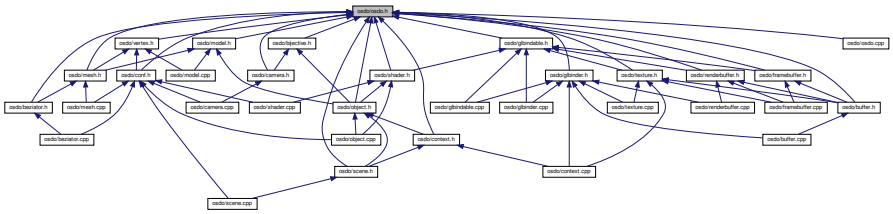
8.54 Файл osdo/osdo.h

#include <cglm/cglm.h>
#include <GL/glew.h>

#include <GL/gl.h>
Діаграма включених заголовочних файлів для osdo.h:



Граф файлів, які включають цей файл:



Макровизначення

- #define UNUSED

Переліки

- enum coord_enum { X = 0 , Y = 1 , Z = 2 }

  Координатні осі.

Функції

- void * operator new[] (size_t size, const char *name, int flags, unsigned debugFlags, const char *file, int line)

  Перевантажений оператор new для EASTL.
- void * operator new[] (size_t size, size_t alignment, size_t alignmentOffset, const char *pName, int flags, unsigned debugFlags, const char *file, int line)

  Перевантажений оператор new для EASTL.

Змінні

- vec3 BASIS0POS

  Початкова позиція камери.

8.54.1 Опис макровизначень

8.54.1.1 UNUSED #define UNUSED

Див. визначення в файлі osdo.h, рядок 25

8.54.2 Опис переліків

8.54.2.1 coord_enum enum coord_enum

Координатні осі.

Елементи переліків

X	Ввісь X
Y	Ввісь Y
Z	Ввісь Z

Див. визначення в файлі osdo.h, рядок 16

8.54.3 Опис функцій

8.54.3.1 operator new[]() [1/2] void* operator new[] (

size_t size,
const char * name,
int flags,
unsigned debugFlags,
const char * file,
int line)

Перевантажений оператор new для EASTL.

Див. визначення в файлі osdo.cpp, рядок 3

8.54.3.2 operator new[]() [2/2]

void* operator new[] (
size_t size,
size_t alignment,
size_t alignmentOffset,
const char * pName,
int flags,
unsigned debugFlags,
const char * file,
int line)

Перевантажений оператор new для EASTL.

Див. визначення в файлі osdo.cpp, рядок 8

8.54.4 Опис змінних

8.54.4.1 BASIS0POS

vec3 BASIS0POS [extern]

Початкова позиція камери.

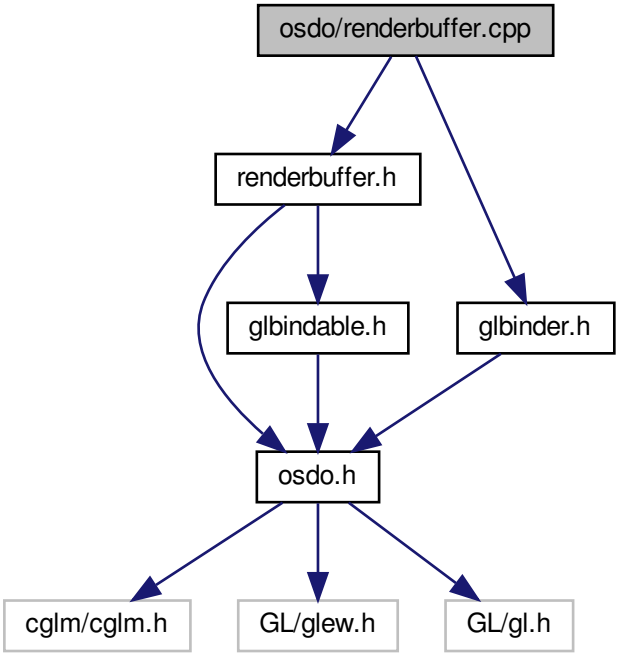
Див. визначення в файлі osdo.cpp, рядок 12

8.55 osdo.h

```
00001 #ifndef OSDO_H
00002 #define OSDO_H
00003
00004 #include <cglm/cglm.h>
00005 #include <GL/glew.h>
00006 #include <GL/gl.h>
00007
00008 /*#define max(a,b) \
00009   ({ __typeof__ (a) _a = (a); \
00010       __typeof__ (b) _b = (b); \
00011     _a > _b ? _a : _b; })*/
00012
00013 /**
00014  * @brief Координатні осі.
00015  */
00016 enum coord_enum {
00017     X = 0, /**< Вісь X */
00018     Y = 1, /**< Вісь Y */
00019     Z = 2, /**< Вісь Z */
00020 };
00021
00022 #ifdef __GNUC__
00023 #define UNUSED __attribute__ ((unused))
00024 #else
00025 #define UNUSED
00026 #endif
00027
00028 /**
00029  * @brief Перевантажений оператор ‘new‘ для EASTL
00030  */
00031 void* operator new[](size_t size, const char* name, int flags,
00032 unsigned debugFlags, const char* file, int line);
00033
00034 /**
00035  * @brief Перевантажений оператор ‘new‘ для EASTL
00036  */
00037 void* operator new[](size_t size, size_t alignment, size_t alignmentOffset,
00038 const char* pName, int flags, unsigned debugFlags, const char* file, int line);
00039
00040 /**
00041  * @brief Початкова позиція камери.
00042  */
00043 extern vec3 BASIS0POS;
00044 //extern vec3 BASIS1POS;
00045 //extern vec3 BASIS2POS;
00046
00047 //extern vec3 BASIS1ROT;
00048 //extern vec3 BASIS2ROT;
00049
00050 #endif // OSDO_H
```

8.56 Файл osdo/renderbuffer.cpp

#include "renderbuffer.h"
#include "glbinder.h"
Діаграма включених заголовочних файлів для renderbuffer.cpp:



8.57 renderbuffer.cpp

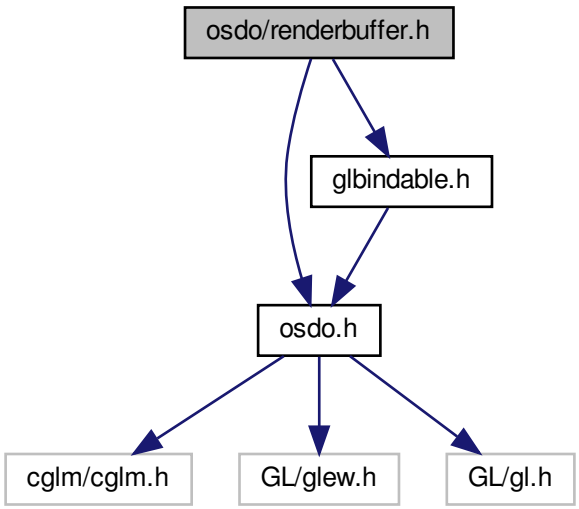
```
00001 #include "renderbuffer.h"
00002 #include "glbinder.h"
00003
00004 GLuint Renderbuffer::_generate() const
00005 {
00006     GLuint id;
00007     glGenRenderbuffers(1, &id);
00008     return id;
00009 }
00010
00011 void Renderbuffer::_bind(const GLuint id, GLenum target) const
00012 {
00013     glBindRenderbuffer(target, id);
00014 }
00015
00016 GLenum Renderbuffer::_default() const
00017 {
00018     return GL_RENDERBUFFER;
00019 }
00020
00021 Renderbuffer::~Renderbuffer() {
00022     glDeleteRenderbuffers(1, &get_id());
00023 }
00024
00025 void Renderbuffer::make_multisample(GLsizei size[2], GLenum target) const {
00026     GlBinder b = binder();
00027     glRenderbufferStorageMultisample(
00028                 GL_RENDERBUFFER, 4, target, size[0], size[1]);
00029 }
00030
```

```
00031 void Renderbuffer::make(GLsizei size[2], GLenum target) const
00032 {
00033     GlBinder b = binder();
00034     glRenderbufferStorage(GL_RENDERBUFFER, target, size[0], size[1]);
00035
00036 }
```

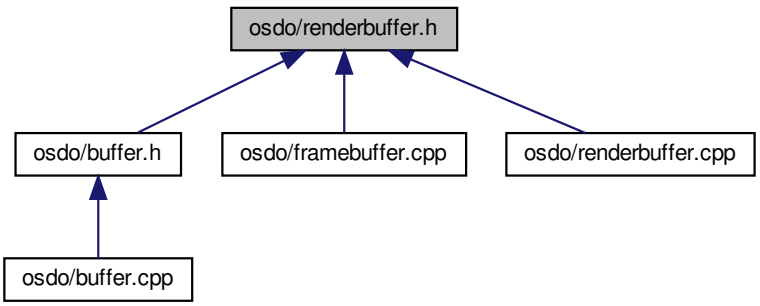
8.58 Файл osdo/renderbuffer.h

Задає клас буфера рендеренгу (для зберігання кольорів або глибини).

#include "osdo.h"
#include "glbindable.h"
Діаграма включених заголовочних файлів для renderbuffer.h:



Граф файлів, які включають цей файл:



Класи

- class Renderbuffer

  Буфер рендеренгу (для зберігання кольорів або глибини)

8.58.1 Детальний опис

Задає клас буфера рендеренгу (для зберігання кольорів або глибини).

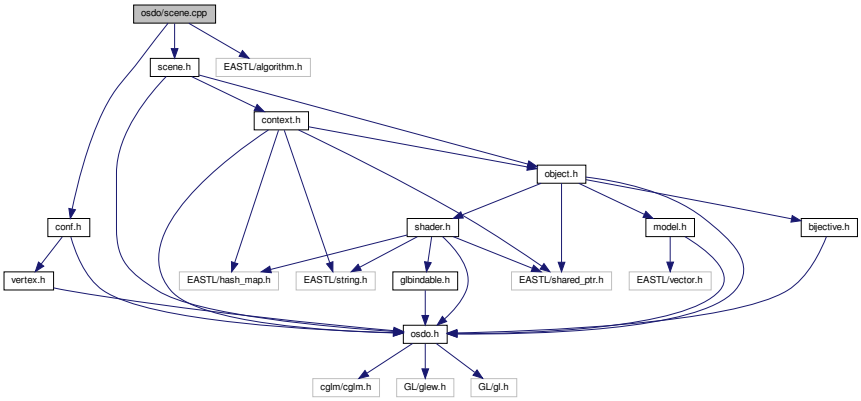
Див. визначення в файлі renderbuffer.h

8.59 renderbuffer.h

```
00001 /**
00002  * @file renderbuffer.h
00003  * @brief Задає клас буфера рендеренгу (для зберігання кольорів або глибини).
00004  */
00005 #ifndef RENDERBUFFER_H
00006 #define RENDERBUFFER_H
00007
00008 #include "osdo.h"
00009 #include "glbindable.h"
00010
00011 /**
00012  * @brief Буфер рендеренгу (для зберігання кольорів або глибини)
00013  */
00014 class Renderbuffer : public GlBindable
00015 {
00016     /**
00017      * @brief Віртуальний метод що створює буфер рендеренгу.
00018      * @return індекс буфер рендеренгу
00019      */
00020     GLuint _generate() const override;
00021     /**
00022      * @brief Віртуальний метод, що прив'язує контекст OpenGL до буферу рендеренгу.
00023      * @param id індекс буфер рендеренгу
00024      * @param target ціль прив'язки буфера рендеренгу
00025      */
00026     virtual void _bind(const GLuint id, GLenum target) const override;
00027     /**
00028      * @brief Віртуальний метод, що задає ціль прив'язки за замовчуванням.
00029      * @return ціль прив'язки за замовчуванням
00030      */
00031     virtual GLenum _default() const override;
00032 public:
00033     /**
00034      * @brief Конструктор, що створює буфер рендеренгу.
00035      */
00036     Renderbuffer() : GlBindable(_generate()) {}
00037     ~Renderbuffer() override;
00038
00039     /**
00040      * @brief Створює буфер рендеренгу з згладжуванням.
00041      * @param size ширина та висота кадру
00042      * @param target ціль буферу рендеренгу
00043      */
00044     void make_multisample(GLsizei size[2], GLenum target) const;
00045
00046     /**
00047      * @brief Створює буфер рендеренгу.
00048      * @param size ширина та висота кадру
00049      * @param target ціль буферу рендеренгу
00050      */
00051     void make(GLsizei size[2], GLenum target) const;
00052 };
00053
00054 #endif // RENDERBUFFER_H
```

8.60 Файл osdo/scene.cpp

#include "scene.h"
#include "conf.h"
#include "EASTL/algorithm.h"
Діаграма включених заголовочних файлів для scene.cpp:



8.61 scene.cpp

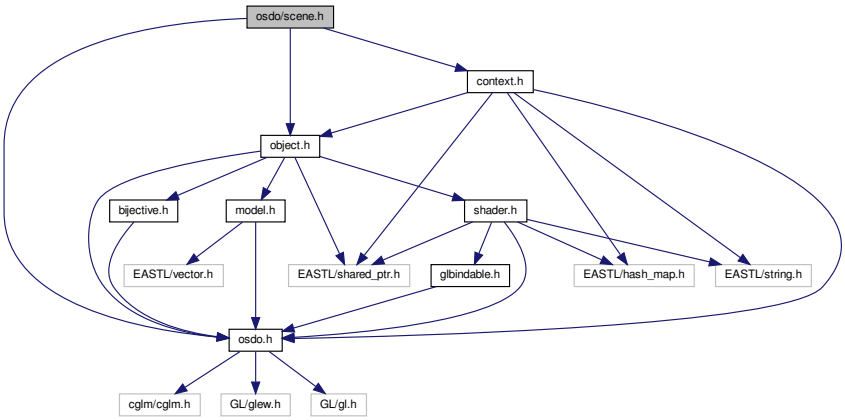
```
00001 #include "scene.h"
00002 #include "conf.h"
00003 #include "EASTL/algorithm.h"
00004 using eastl::transform;
00005 using eastl::make_shared;
00006
00007 Scene::Scene(const Context::Models &objects) : objects(objects) {
00008 }
00009
00010 shared_ptr<Scene> Scene::create(const Context::Models &objects)
00011 {
00012     return make_shared<Scene>(objects);
00013 }
```

8.62 Файл osdo/scene.h

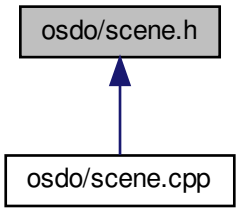
Задає клас сцени із об'єктами.

#include "osdo.h"
#include "object.h"
#include "context.h"

Діаграма включених заголовочних файлів для scene.h:



Граф файлів, які включають цей файл:



Класи

- struct Scene

  Сцена із об'єктами.

8.62.1 Детальний опис

Задає клас сцени із об'єктами.

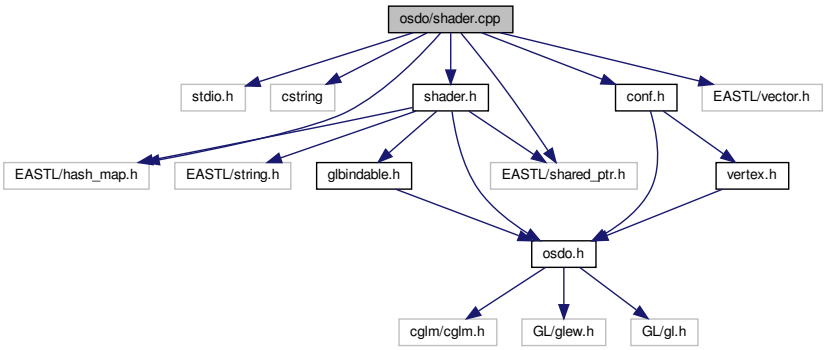
Див. визначення в файлі scene.h

8.63 scene.h

```
00001 /**
00002  * @file scene.h
00003  * @brief Задає клас сцени із об'єктами.
00004  */
00005 #ifndef SCENE_H
00006 #define SCENE_H
00007
00008 #include "osdo.h"
00009
00010 #include "object.h"
00011 #include "context.h"
00012
00013 /**
00014  * @brief Сцена із об'єктами.
00015  */
00016 struct Scene {
00017     /**
00018      * @brief Об'єкти у сцені.
00019      */
00020     hash_map<string, Object> objects;
00021
00022     /**
00023      * @brief Конструктор, що створює об'єкти у сцені за заготовленими у агрументі 'objects'
00024      * @param objects заготовлені об'єкти для додавання у сцену
00025      */
00026     Scene(const Context::Models& objects);
00027
00028     /**
00029      * @brief Створює сцену
00030      * @param objects заготовлені об'єкти для додавання у сцену
00031      * @return Розумний вказівник на об'єкт сцени.
00032      */
00033     static shared_ptr<Scene> create(const Context::Models& objects);
00034 };
00035
00036 #endif // SCENE_H
```

8.64 Файл osdo/shader.cpp

#include <stdio.h>
#include <cstring>
#include "shader.h"
#include "conf.h"
#include "EASTL/hash_map.h"
#include "EASTL/shared_ptr.h"
#include "EASTL/vector.h"
Діаграма включених заголовочних файлів для shader.cpp:



Класи

- class ShaderSource

Функції

- char * readFromFile (const char *path)
- bool check_shader (GLuint shader, const int type)
- shared_ptr< Shader > compile (vector< shared_ptr< ShaderSource >> shaders)

8.64.1 Опис функцій

8.64.1.1 check_shader() bool check_shader (GLuint shader, const int type)

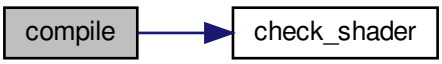
Див. визначення в файлі shader.cpp, рядок 40

Граф викликів для цієї функції:

8.64.1.2 compile() shared_ptr<Shader> compile (vector< shared_ptr< ShaderSource >> shaders)

Див. визначення в файлі shader.cpp, рядок 95

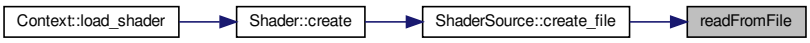
Граф всіх викликів цієї функції:



8.64.1.3 readFromFile() char* readFromFile (const char * path)

Див. визначення в файлі shader.cpp, рядок 21

Граф викликів для цієї функції:



8.65 shader.cpp

```
00001 #include <stdio.h>
00002 #include <cstring>
00003
00004 #include "shader.h"
00005 #include "conf.h"
00006 #include "EASTL/hash_map.h"
00007 #include "EASTL/shared_ptr.h"
00008 #include "EASTL/vector.h"
00009 using eastl::hash_map;
00010 using eastl::make_shared;
00011 using eastl::vector;
00012
00013 static hash_map<ShaderType, GLenum> TYPES_MAP = {
00014     {VERT_SHADER, GL_VERTEX_SHADER},
00015     {TESC_SHADER, GL_TESS_CONTROL_SHADER},
00016     {TESE_SHADER, GL_TESS_EVALUATION_SHADER},
00017     {GEOM_SHADER, GL_GEOMETRY_SHADER},
00018     {FRAG_SHADER, GL_FRAGMENT_SHADER},
00019 };
00020
00021 char * readFromFile(const char *path) {
00022     char* data;
00023     size_t size;
00024     FILE *file = fopen(path, "r");
00025     if (file == nullptr) {
00026         printf("ERROR: failed to open file %s\n", path);
00027         return nullptr;
00028     }
00029     fseek(file, 0L, SEEK_END);
00030     size = static_cast<size_t>(ftell(file));
00031     fseek(file, 0L, SEEK_SET);
00032     data = static_cast<char*>(malloc(size + 1));
00033     fread(data, 1, size, file);
00034     data[size] = 0;
00035     fclose(file);
00036     return data;
00037 }
00038
00039 // utility function for checking shader compilation/linking errors.
00040 bool check_shader(GLuint shader, const int type) {
00041     GLint status = 0, size = 0;
00042     GLchar *log;
00043     GLuint status_type = GL_COMPILE_STATUS;
00044     void (*gl_get)(GLuint, GLuint, GLint*) = glGetShaderiv;
00045     void (*gl_info)(GLuint, GLint, GLint*, GLchar*) = glGetShaderInfoLog;
00046
00047     if (type == 0) {
00048         gl_get = glGetProgramiv;
00049         status_type = GL_LINK_STATUS;
00050         gl_info = glGetProgramInfoLog;
00051     }
00052
00053     gl_get(shader, status_type, &status);
00054     if (status == GL_FALSE) {
00055         gl_get(shader, GL_INFO_LOG_LENGTH, &size);
00056         log = static_cast<GLchar*>(malloc(static_cast<size_t>(size)));
00057         if (log == nullptr) {
00058             printf("Got some error, but cant allocate memory to read it.\n");
00059             return false;
00060         }
00061         gl_info(shader, size, &size, log);
00062         puts(log);
```

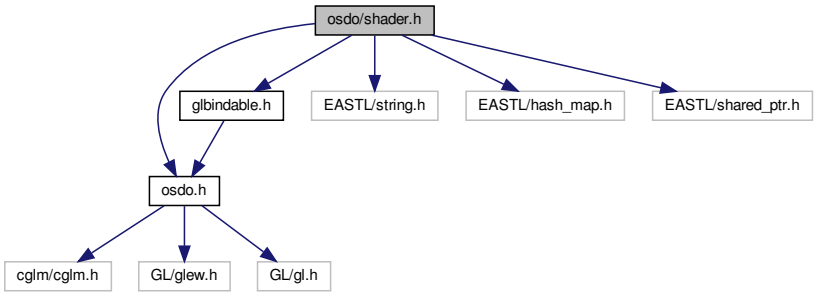
```
00063         fflush(stdout);
00064         free(log);
00065         return false;
00066     }
00067     return true;
00068 }
00069
00070 class ShaderSource {
00071     const GLuint id;
00072 public:
00073     ShaderSource(const GLuint id) : id(id) {}
00074     static shared_ptr<ShaderSource> create(GLenum type, const char *code) {
00075         const GLuint shader = glCreateShader(type);
00076         glShaderSource(shader, 1, &code, nullptr);
00077         glCompileShader(shader);
00078         if (!check_shader(shader, 1)) {
00079             return {};
00080         }
00081         return make_shared<ShaderSource>(shader);
00082     }
00083     static shared_ptr<ShaderSource> create_file(GLenum type, const string& path) {
00084         GLchar* code = readFromFile(path.c_str());
00085         if (!code)
00086             return {};
00087         return create(type, code);
00088     }
00089     GLuint get_id() {return id;}
00090     void attach(const GLuint program) {
00091         glAttachShader(program, id);
00092     }
00093 };
00094
00095 shared_ptr<Shader> compile(vector<shared_ptr<ShaderSource» shaders) {
00096     for (auto &i : shaders) {
00097         if (!i)
00098             return {};
00099     }
00100
00101     GLuint sh = glCreateProgram();
00102     for (auto &i : shaders) {
00103         i->attach(sh);
00104     }
00105     glLinkProgram(sh);
00106     if (!check_shader(sh, 0)) {
00107         return {};
00108     }
00109
00110     return make_shared<Shader>(sh);
00111 }
00112
00113 void Shader::_bind(const GLuint id, UNUSED GLenum target) const
00114 {
00115     glUseProgram(id);
00116 }
00117
00118 Shader::Shader(const GLuint shader) : GlBindable(shader) {}
00119
00120 Shader::~Shader() {
00121     glDeleteProgram(this->get_id());
00122 }
00123
00124 shared_ptr<Shader> Shader::create(const Shader::shader_map& shaders_paths)
00125 {
00126     vector<shared_ptr<ShaderSource» shaders;
00127     for (auto& i : shaders_paths) {
00128         shaders.push_back(
00129                 ShaderSource::create_file(TYPES_MAP[i.first], i.second));
00130     }
00131     return compile(shaders);
00132 }
00133
00134 void Shader::set_bool(const char* name, bool value) {
00135     glUniform1i(glGetUniformLocation(this->get_id(), name), static_cast<int>(value));
00136 }
00137
00138 void Shader::set_int(const char* name, int value) {
00139     glUniform1i(glGetUniformLocation(this->get_id(), name), value);
00140 }
00141
00142 void Shader::set_float(const char* name, float value) {
00143     glUniform1f(glGetUniformLocation(this->get_id(), name), value);
00144 }
00145
00146 void Shader::set_vec2(const char* name, vec2 value) {
00147     glUniform2fv(glGetUniformLocation(this->get_id(), name),
00148                  1, &value[0]);
00149 }
```

```
00150
00151 void Shader::set_vec2f(const char* name,
00152                         float x, float y) {
00153     glUniform2f(glGetUniformLocation(this->get_id(), name), x, y);
00154 }
00155
00156 void Shader::set_vec3(const char* name, vec3 value) {
00157     glUniform3fv(glGetUniformLocation(this->get_id(), name),
00158                  1, &value[0]);
00159 }
00160
00161 void Shader::set_vec3f(const char* name,
00162                         float x, float y, float z) {
00163     glUniform3f(glGetUniformLocation(this->get_id(), name), x, y, z);
00164 }
00165
00166 void Shader::set_vec4(const char* name, vec4 value) {
00167     glUniform4fv(glGetUniformLocation(this->get_id(), name),
00168                  1, &value[0]);
00169 }
00170
00171 void Shader::set_vec4f(const char* name,
00172                         float x, float y, float z, float w) {
00173     glUniform4f(glGetUniformLocation(this->get_id(), name), x, y, z, w);
00174 }
00175
00176 void Shader::set_mat2(const char* name, mat2 mat) {
00177     glUniformMatrix2fv(glGetUniformLocation(this->get_id(), name),
00178                        1, GL_FALSE, &mat[0][0]);
00179 }
00180
00181 void Shader::set_mat3(const char* name, mat3 mat) {
00182     glUniformMatrix3fv(glGetUniformLocation(this->get_id(), name),
00183                        1, GL_FALSE, &mat[0][0]);
00184 }
00185
00186 void Shader::set_mat4(const char* name, mat4 mat) {
00187     glUniformMatrix4fv(glGetUniformLocation(this->get_id(), name),
00188                        1, GL_FALSE, &mat[0][0]);
00189 }
```

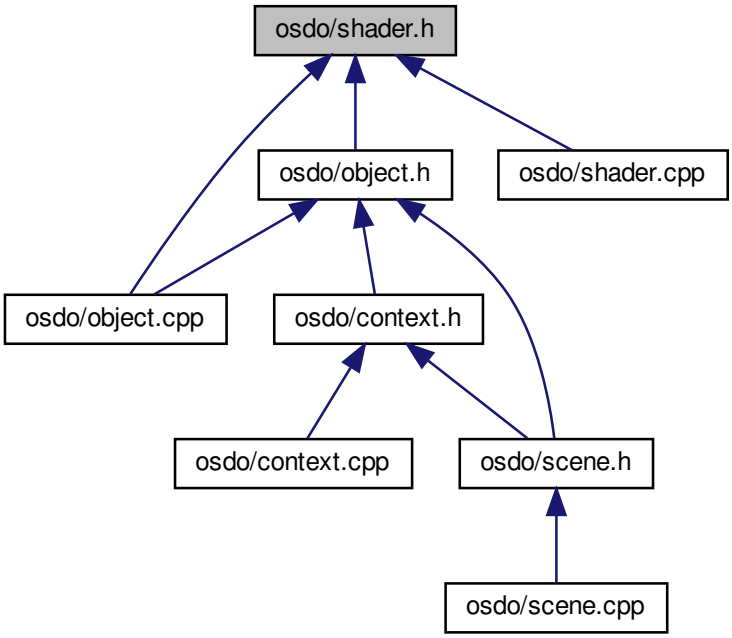
8.66 Файл osdo/shader.h

Задає клас шейдеру.

#include "osdo.h"
#include "glbindable.h"
#include "EASTL/string.h"
#include "EASTL/hash_map.h"
#include "EASTL/shared_ptr.h"
Діаграма включених заголовочних файлів для shader.h:



Граф файлів, які включають цей файл:



Класи

- class Shader

  Клас взаємодії з шейдером у видеокарті.

Переліки

- enum ShaderType {
  VERT_SHADER , TESC_SHADER , TESE_SHADER , GEOM_SHADER ,
  FRAG_SHADER }

  Тип шейдеру

8.66.1 Детальний опис

Задає клас шейдеру.

Див. визначення в файлі shader.h

8.66.2 Опис переліків

8.66.2.1 ShaderType enum ShaderType

Тип шейдеру

Елементи переліків

VERT_SHADER	вершинний шейдер
TESC_SHADER	теселяційний контрольний шейдер
TESE_SHADER	теселяційний обчислювальний шейдер
GEOM_SHADER	геометричний шейдер
FRAG_SHADER	фрагмнтний шейдер

Див. визначення в файлі shader.h, рядок 20

8.67 shader.h

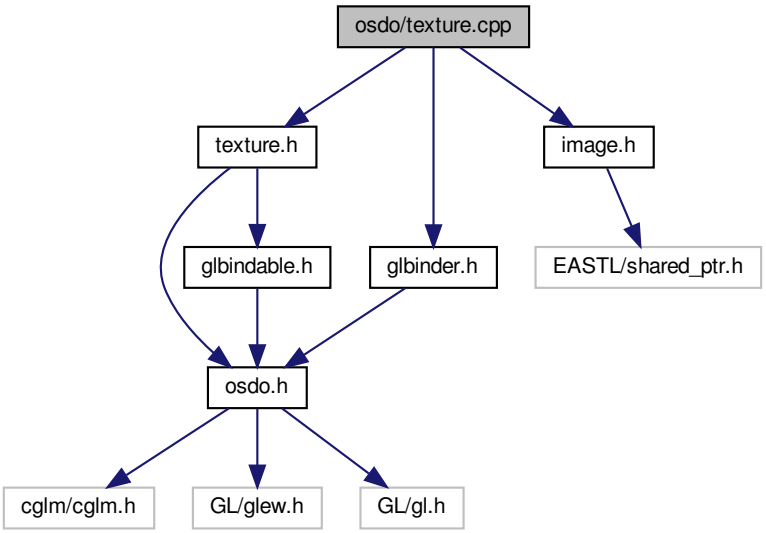
```
00001 /**
00002  * @file shader.h
00003  * @brief Задає клас шейдеру.
00004  */
00005 #ifndef SHADER_H
00006 #define SHADER_H
00007
00008 #include "osdo.h"
00009 #include "glbindable.h"
00010 #include "EASTL/string.h"
00011 #include "EASTL/hash_map.h"
00012 #include "EASTL/shared_ptr.h"
00013 using eastl::string;
00014 using eastl::hash_map;
00015 using eastl::shared_ptr;
00016
00017 /**
00018  * @brief Тип шейдеру
00019  */
00020 enum ShaderType {
00021     VERT_SHADER, /**< вершинний шейдер */
00022     TESC_SHADER, /**< теселяційний контрольний шейдер */
00023     TESE_SHADER, /**< теселяційний обчислювальний шейдер */
00024     GEOM_SHADER, /**< геометричний шейдер */
00025     FRAG_SHADER, /**< фрагмнтний шейдер */
00026 };
00027
00028 /**
00029  * @brief Клас взаємодії з шейдером у видеокарті.
00030  */
00031 class Shader : public GlBindable {
00032     /**
00033      * @brief Віртуальний метод, що прив'язує контекст OpenGL до шейдеру.
00034      * @param id індекс шейдеру OpenGL
00035      * @param target не використовується, реалізован для інтерфейсу 'GlBindable::_bind'
00036      */
00037     virtual void _bind(const GLuint id, GLenum target) const override;
00038 public:
00039     /**
00040      * @brief тип для зберігання шляхів файлів шейдеру по їх типам.
00041      */
00042     typedef hash_map<ShaderType, string> shader_map;
00043     Shader(const GLuint shader);
00044     ~Shader() override;
00045
00046     /**
00047      * @brief Створює об'єкт шейдеру за заданими шляхами файлів шейдерів.
00048      * @param shaders_paths шляхи файлів шейдеру по їх типам
00049      * @return розумний вказівник на об'єкт шейдеру
00050      */
00051     static shared_ptr<Shader> create(const shader_map& shaders_paths);
00052
00053     /**
00054      * @brief Задати поле шейдеру типу 'bool'
00055      * @param name ім'я поля
00056      * @param value значення поля
00057      */
00058     void set_bool (const char* name, bool value);
00059     /**
00060      * @brief Задати поле шейдеру типу 'int'
00061      * @param name ім'я поля
00062      * @param value значення поля
00063      */
00064     void set_int  (const char* name, int value);
00065     /**
```

```
00066     * @brief Задати поле шейдеру типу ‘float‘
00067     * @param name ім’я поля
00068     * @param value значення поля
00069     */
00070    void set_float(const char* name, float value);
00071    /**
00072     * @brief Задати поле шейдеру типу ‘vec2‘
00073     * @param name ім’я поля
00074     * @param value значення поля
00075     */
00076    void set_vec2 (const char* name, vec2 value);
00077    /**
00078     * @brief Задати вектор-поле шейдеру типу ‘vec2‘
00079     * @param name ім’я поля
00080     * @param x значення першого елементу вектор-поля
00081     * @param y значення другого елементу вектор-поля
00082     */
00083    void set_vec2f(const char* name, float x, float y);
00084    /**
00085     * @brief Задати поле шейдеру типу ‘vec3‘
00086     * @param name ім’я поля
00087     * @param value значення поля
00088     */
00089    void set_vec3 (const char* name, vec3 value);
00090    /**
00091     * @brief Задати вектор-поле шейдеру типу ‘vec3‘
00092     * @param name ім’я поля
00093     * @param x значення першого елементу вектор-поля
00094     * @param y значення другого елементу вектор-поля
00095     * @param z значення третього елементу вектор-поля
00096     */
00097    void set_vec3f(const char* name, float x, float y, float z);
00098    /**
00099     * @brief Задати поле шейдеру типу ‘vec4‘
00100     * @param name ім’я поля
00101     * @param value значення поля
00102     */
00103    void set_vec4 (const char* name, vec4 value);
00104    /**
00105     * @brief Задати вектор-поле шейдеру типу ‘vec4‘
00106     * @param name ім’я поля
00107     * @param x значення першого елементу вектор-поля
00108     * @param y значення другого елементу вектор-поля
00109     * @param z значення третього елементу вектор-поля
00110     * @param w значення четвертого елементу вектор-поля
00111     */
00112    void set_vec4f(const char* name, float x, float y, float z, float w);
00113    /**
00114     * @brief Задати поле шейдеру типу ‘mat2‘
00115     * @param name ім’я поля
00116     * @param mat значення поля
00117     */
00118    void set_mat2 (const char* name, mat2 mat);
00119    /**
00120     * @brief Задати поле шейдеру типу ‘mat3‘
00121     * @param name ім’я поля
00122     * @param mat значення поля
00123     */
00124    void set_mat3 (const char* name, mat3 mat);
00125    /**
00126     * @brief Задати поле шейдеру типу ‘mat4‘
00127     * @param name ім’я поля
00128     * @param mat значення поля
00129     */
00130    void set_mat4 (const char* name, mat4 mat);
00131 };
00132
00133 #endif // SHADER_H
```

8.68 Файл osdo/texture.cpp

```
#include "texture.h"
#include "glbinder.h"
#include "image.h"
```

Діаграма включених заголовочних файлів для texture.cpp:



Функції

- void LoadTextureFromFile (const pixel_t(*data)[], int width, int height, const GLuint id)

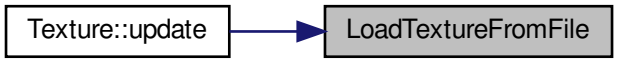
8.68.1 Опис функцій

8.68.1.1 LoadTextureFromFile()

void LoadTextureFromFile (
const pixel_t(*) data[],
int width,
int height,
const GLuint id)

Див. визначення в файлі texture.cpp, рядок 26

Граф викликів для цієї функції:



8.69 texture.cpp

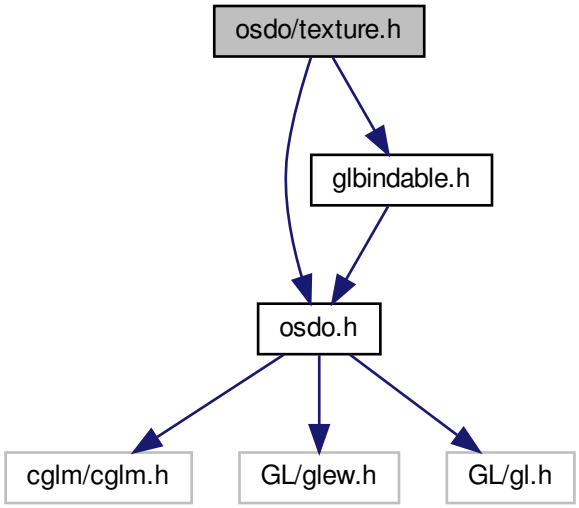
```
00001 #include "texture.h"
00002 #include "glbinder.h"
00003 #include "image.h"
00004
00005 GLuint Texture::_generate() const
00006 {
00007     GLuint id;
00008     glGenTextures(1, &id);
00009     return id;
00010 }
00011
00012 void Texture::_bind(const GLuint id, GLenum target) const
00013 {
00014     glBindTexture(target, id);
00015 }
00016
00017 GLenum Texture::_default() const
00018 {
00019     return GL_TEXTURE_2D_MULTISAMPLE;
00020 }
00021
00022 Texture::~Texture() {
00023     glDeleteTextures(1, &get_id());
00024 }
00025
00026 void LoadTextureFromFile(const pixel_t (*data)[], int width, int height,
00027                          const GLuint id)
00028 {
00029     glBindTexture(GL_TEXTURE_2D, id);
00030
00031     glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MIN_FILTER, GL_LINEAR);
00032     glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MAG_FILTER, GL_LINEAR);
00033     glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_WRAP_S, GL_CLAMP_TO_EDGE);
00034     glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_WRAP_T, GL_CLAMP_TO_EDGE);
00035
00036 #if defined(GL_UNPACK_ROW_LENGTH) && !defined(__EMSCRIPTEN__)
00037     glPixelStorei(GL_UNPACK_ROW_LENGTH, 0);
00038 #endif
00039     glTexImage2D(GL_TEXTURE_2D, 0, GL_RGBA, width, height, 0, GL_RGBA,
00040                  GL_UNSIGNED_BYTE, data);
00041 }
00042
00043 void Texture::update(const Image &image) const {
00044     LoadTextureFromFile(image.data.get(), image.width, image.height, get_id());
00045 }
00046
00047 void Texture::make_2d_multisample(GLsizei size[]) const {
00048     GlBinder b = binder(GL_TEXTURE_2D_MULTISAMPLE);
00049     glTexImage2DMultisample(GL_TEXTURE_2D_MULTISAMPLE, 4, GL_RGB,
00050                             size[0], size[1], GL_TRUE);
00051 }
00052
00053 void Texture::make_2d(GLsizei size[]) const {
00054     GlBinder b = binder(GL_TEXTURE_2D);
00055     glTexImage2D(GL_TEXTURE_2D, 0, GL_RGB, size[0], size[1], 0, GL_RGB,
00056             GL_UNSIGNED_BYTE, nullptr);
00057     glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MIN_FILTER, GL_NEAREST);
00058     glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MAG_FILTER, GL_NEAREST);
00059 }
```

8.70 Файл osdo/texture.h

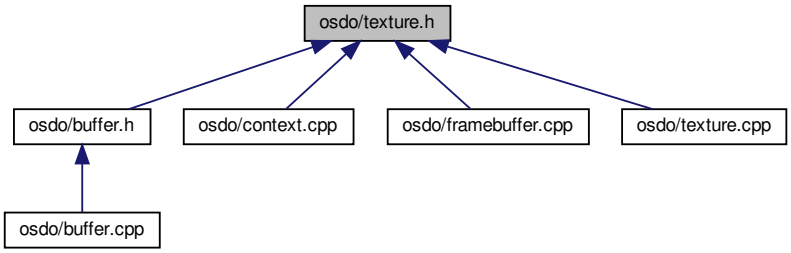
Задає клас текстури.

#include "osdo.h"
#include "glbindable.h"

Діаграма включених заголовочних файлів для texture.h:



Граф файлів, які включають цей файл:



Класи

- class Texture

  Клас текстури, що зберігаэться у відеокарті.

8.70.1 Детальний опис

Задає клас текстури.

Див. визначення в файлі texture.h

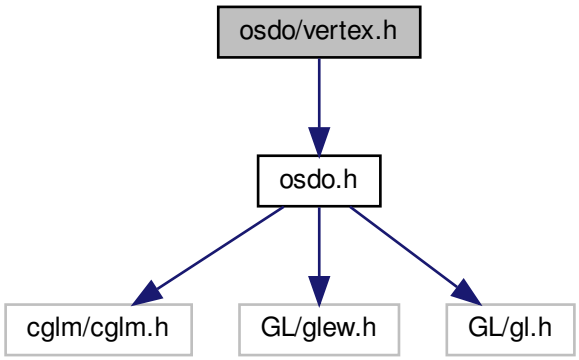
8.71 texture.h

```
00001 /**
00002  * @file texture.h
00003  * @brief Задає клас текстури.
00004  */
00005 #ifndef TEXTURE_H
00006 #define TEXTURE_H
00007
00008 #include "osdo.h"
00009 #include "glbindable.h"
00010
00011 class Image;
00012
00013 /**
00014  * @brief Клас текстури, що зберігаэться у відеокарті.
00015  */
00016 class Texture : public GlBindable
00017 {
00018     /**
00019      * @brief Віртуальний метод що створює текстуру.
00020      * @return індекс текстури
00021      */
00022     GLuint _generate() const override;
00023     /**
00024      * @brief Віртуальний метод, що прив'язує контекст OpenGL до текстури.
00025      * @param id індекс текстури
00026      * @param target ціль прив'язки текстури
00027      */
00028     virtual void _bind(const GLuint id, GLenum target) const override;
00029     /**
00030      * @brief Віртуальний метод, що задає ціль прив'язки за замовчуванням.
00031      * @return ціль прив'язки за замовчуванням
00032      */
00033     virtual GLenum _default() const override;
00034 public:
00035     Texture() : GlBindable(_generate()) {}
00036     ~Texture() override;
00037
00038     /**
00039      * @brief Завантажує зображення у текстуру.
00040      * @param image зображення
00041      */
00042     void update(const Image& image) const;
00043
00044     /**
00045      * @brief Створює текстуру з згладжуванням.
00046      * @param size ширина та висота кадру
00047      */
00048     void make_2d_multisample(GLsizei size[2]) const;
00049
00050     /**
00051      * @brief Створює текстуру.
00052      * @param size ширина та висота кадру
00053      */
00054     void make_2d(GLsizei size[2]) const;
00055 };
00056
00057 #endif // TEXTURE_H
```

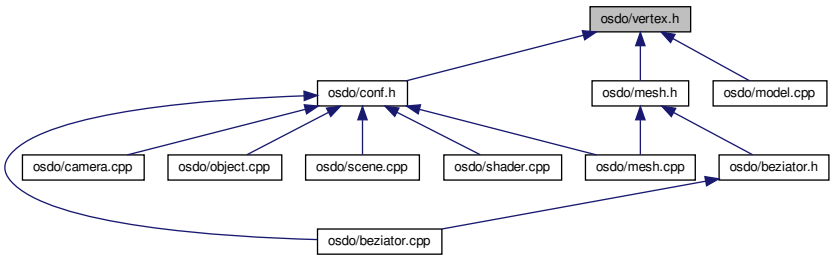
8.72 Файл osdo/vertex.h

Задає структуру вершини.

#include "osdo.h"
Діаграма включених заголовочних файлів для vertex.h:



Граф файлів, які включають цей файл:



Класи

- struct Vertex

  Структура вершини, для передачі у відеокарту.

8.72.1 Детальний опис

Задає структуру вершини.

Див. визначення в файлі vertex.h

8.73 vertex.h

```
00001 /**
00002  * @file vertex.h
00003  * @brief Задає структуру вершини.
00004  */
00005 #ifndef VERTEX_H
00006 #define VERTEX_H
00007 #include "osdo.h"
00008
00009 /**
00010  * @brief Структура вершини, для передачі у відеокарту.
00011  */
00012 struct Vertex {
00013     /**
00014      * @brief Позиція вершини у просторі.
00015      */
00016     vec4 position;
00017     /**
00018      * @brief Нормаль вершини.
00019      */
00020     vec3 normal;
00021     /**
00022      * @brief Колір вершини.
00023      */
00024     unsigned char color[4];
00025     /**
00026      * @brief Координати вершини на текстурі.
00027      */
00028     vec2 uv;
00029 };
00030
00031 #endif // VERTEX_H
```

8.74 Файл res/bezier.frag

8.75 bezier.frag

```
00001 #version 420 core
00002 layout(location = 0) out vec4 FragColor;
00003
00004 struct Data {
00005     vec4 color;
00006     vec2 uv;
00007     vec3 normal;
00008     vec3 frag_pos;
00009 };
00010
00011 layout(location = 0) in Data data;
00012
00013 struct DirLight {
00014     vec3 direction;
00015
00016     vec3 ambient;
00017     vec3 diffuse;
00018     vec3 specular;
00019 };
00020
00021 uniform vec3 viewPos;
00022 uniform DirLight dirLight;
00023 uniform float materialShininess;
00024 uniform float alpha;
00025 uniform bool textured;
00026 uniform sampler2D textureSample;
00027
00028 // calculates the color when using a directional light.
00029 vec3 CalcDirLight(DirLight light, vec3 normal, vec3 viewDir, vec3 color)
00030 {
00031     vec3 lightDir = normalize(-light.direction);
00032     // diffuse shading
00033     float diff = max(dot(normal, lightDir), 0.0);
00034     // specular shading
00035     vec3 reflectDir = reflect(-lightDir, normal);
00036     float spec = pow(max(dot(viewDir, reflectDir), 0.0), materialShininess);
00037     // combine results
00038     vec3 ambient = light.ambient * color;
00039     vec3 diffuse = light.diffuse * diff * color;
00040     vec3 specular = light.specular * spec * color;
00041     return (ambient + diffuse + specular);
00042 }
00043
00044 void main()
00045 {
```

```
00046     vec3 norm = normalize(data.normal);
00047     vec3 viewDir = normalize(-viewPos - data.frag_pos);
00048     vec4 color = data.color;
00049     if (textured) {
00050         color = texture(textureSample, data.uv);
00051     }
00052     vec3 tmp = CalcDirLight(dirLight, norm, viewDir, vec3(color));
00053     FragColor = vec4(tmp, alpha);
00054 }
```

8.76 Файл res/bezier.geom

8.77 bezier.geom

```
00001 #version 420 core
00002 layout(triangles) in;
00003 layout(triangle_strip, max_vertices=16) out;
00004
00005 struct Data {
00006     vec4 color;
00007     vec2 uv;
00008     vec3 normal;
00009     vec3 frag_pos;
00010 };
00011
00012 in Data vertex[3];
00013 out Data geometry;
00014
00015 void main() {
00016     int i;
00017     for(i = 0; i < 16; i++) {
00018         gl_Position = gl_in[i].gl_Position;
00019         geometry.color = vertex[i].color;
00020         geometry.uv = vertex[i].uv;
00021         geometry.pos = vertex[i].pos;
00022         geometry.normal = vertex[i].normal;
00023         EmitVertex();
00024     }
00025     EndPrimitive();
00026 }
```

8.78 Файл res/bezier.tesc

8.79 bezier.tesc

```
00001 #version 420 core
00002
00003 struct Data {
00004     vec4 color;
00005     vec2 uv;
00006     vec3 normal;
00007     vec3 frag_pos;
00008 };
00009
00010 layout(location = 0) in Data inData[];
00011 layout(location = 0) out Data outData[];
00012
00013 uniform int inner;
00014 uniform int outer;
00015
00016 layout(vertices = 16) out;
00017
00018 void main(void) {
00019     gl_TessLevelInner[0] = inner;
00020     gl_TessLevelInner[1] = inner;
00021     gl_TessLevelOuter[0] = outer;
00022     gl_TessLevelOuter[1] = outer;
00023     gl_TessLevelOuter[2] = outer;
00024     gl_TessLevelOuter[3] = outer;
00025
00026     gl_out[gl_InvocationID].gl_Position = gl_in[gl_InvocationID].gl_Position;
00027     outData[gl_InvocationID].color = inData[gl_InvocationID].color;
00028     outData[gl_InvocationID].uv = inData[gl_InvocationID].uv;
00029     outData[gl_InvocationID].normal = inData[gl_InvocationID].normal;
00030     outData[gl_InvocationID].frag_pos = inData[gl_InvocationID].frag_pos;
00031 }
```

8.80 Файл res/bezier.tese

8.81 bezier.tese

```
00001 #version 420 core
00002
00003 layout(quads, equal_spacing) in;
00004
00005 struct Data {
00006     vec4 color;
00007     vec2 uv;
00008     vec3 normal;
00009     vec3 frag_pos;
00010 };
00011
00012 layout(location = 0) in Data inData[];
00013 layout(location = 0) out Data outData;
00014
00015 mat4  b = mat4 ( 1,  0,  0, 0,
00016                 -3,  3,  0, 0,
00017                  3, -6,  3, 0,
00018                 -1,  3, -3, 1);
00019
00020 void main(void) {
00021     float x = gl_TessCoord.x;
00022     float y = gl_TessCoord.y;
00023     vec4 u = vec4 (1.0, x, x*x, x*x*x);
00024     vec4 v = vec4 (1.0, y, y*y, y*y*y);
00025     vec4 uu = vec4 (0, 1.0, 2*x, 3*x*x);
00026     vec4 vv = vec4 (0, 1.0, 2*y, 3*y*y);
00027
00028     vec4 bu = b * u;
00029     vec4 bv = b * v;
00030     vec4 buu = b * uu;
00031     vec4 bvv = b * vv;
00032
00033     mat4 pu[4], pv[4], cu, cv;
00034     for (int i = 0; i < 4; i++) {
00035         for (int j = 0; j < 4; j++) {
00036             pv[i][j] = gl_in[j*4 + i].gl_Position;
00037         }
00038     }
00039     for (int i = 0; i < 4; i++) {
00040         cv[i] = pv[i] * bv;
00041     }
00042
00043     gl_Position = cv * bu;
00044
00045     for (int i = 0; i < 4; i++) {
00046         for (int j = 0; j < 4; j++) {
00047             pu[i][j] = vec4(inData[i*4 + j].normal, 1);
00048             pv[i][j] = vec4(inData[j*4 + i].normal, 1);
00049         }
00050     }
00051     for (int i = 0; i < 4; i++) {
00052         cu[i] = pu[i] * bu;
00053         cv[i] = pv[i] * bv;
00054     }
00055     vec4 du = cv * buu, dv = cu * bvv;
00056     outData.normal = cross(vec3(du), vec3(dv));
00057
00058     for (int i = 0; i < 4; i++) {
00059         for (int j = 0; j < 4; j++) {
00060             pv[i][j] = vec4(inData[j*4 + i].frag_pos, 1);
00061         }
00062     }
00063     for (int i = 0; i < 4; i++) {
00064         cv[i] = pv[i] * bv;
00065     }
00066     outData.frag_pos = vec3(cv * bu);
00067
00068
00069     /*for (int i = 0; i < 4; i++) {
00070         for (int j = 0; j < 4; j++) {
00071             pv[i][j] = vec4(inData[i*4 + i].uv, 0, 1);
00072         }
00073     }
00074     for (int i = 0; i < 4; i++) {
00075         cv[i] = pv[i] * bv;
00076     }
00077     outData.uv = vec2(cv * bu);*/
00078     outData.uv = vec2(x, y);
00079
00080     outData.color = inData[0].color;
00081 }
```

8.82 Файл res/bezier.vert

8.83 bezier.vert

```
00001 #version 420 core
00002 layout (location = 0) in vec3 position;
00003 layout (location = 1) in vec3 normal;
00004 layout (location = 2) in vec4 color;
00005 layout (location = 3) in vec2 uv;
00006
00007 struct Data {
00008     vec4 color;
00009     vec2 uv;
00010     vec3 normal;
00011     vec3 frag_pos;
00012 };
00013
00014 layout(location = 0) out Data data;
00015
00016 uniform mat4 model;
00017 uniform mat4 camera;
00018 uniform mat4 projection;
00019
00020 void main()
00021 {
00022     mat4 trans = projection * camera * model;
00023     vec4 pos = trans * vec4(position, 1.0);
00024     gl_Position = pos;
00025     data.color = color;
00026     data.uv = uv;
00027     data.frag_pos = vec3(model * vec4(position, 1.0));
00028     data.normal = mat3(transpose(inverse(model))) * vec3(normal);
00029 }
```