

Компьютерная графика

Лекция 15: рендеринг текста, bitmap-шрифты, векторные шрифты, (M)SDF-шрифты, чем заняться дальше

2023

- Текст – очень сложная штука

Рендеринг текста

- Текст – очень сложная штука
- Слева направо (латиница, кириллица), справа налево (арабский, иврит), сверху вниз (хангиль, старомонгольское письмо)

Рендеринг текста

- Текст – очень сложная штука
- Слева направо (латиница, кириллица), справа налево (арабский, иврит), сверху вниз (хангиль, старомонгольское письмо)
- Иероглифы (кандзи), слоговое письмо (катакана, хирагана), консонантное письмо (арабский), консонантно-вокалическое письмо (латиница, кириллица)

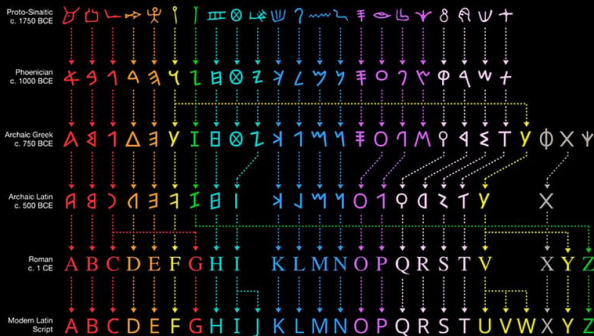
Рендеринг текста

- Текст – очень сложная штука
- Слева направо (латиница, кириллица), справа налево (арабский, иврит), сверху вниз (хангиль, старомонгольское письмо)
- Иероглифы (кандзи), слоговое письмо (катакана, хирагана), консонантное письмо (арабский), консонантно-вокалическое письмо (латиница, кириллица)
- Одна *графема* может представлять один или несколько звуков или слогов

Рендеринг текста

- Текст – очень сложная штука
- Слева направо (латиница, кириллица), справа налево (арабский, иврит), сверху вниз (хангиль, старомонгольское письмо)
- Иероглифы (кандзи), слоговое письмо (катакана, хирагана), консонантное письмо (арабский), консонантно-вокалическое письмо (латиница, кириллица)
- Одна *графема* может представлять один или несколько звуков или слогов
- Могут быть сложные правила по соединению символов между собой (арабский, лигатуры в латинице), дополнения к символам (диакритика)

Evolution of the Alphabet



By Matt Baker | UsefulCharts.com
Reproduction for personal use only. All rights reserved.

Sample text

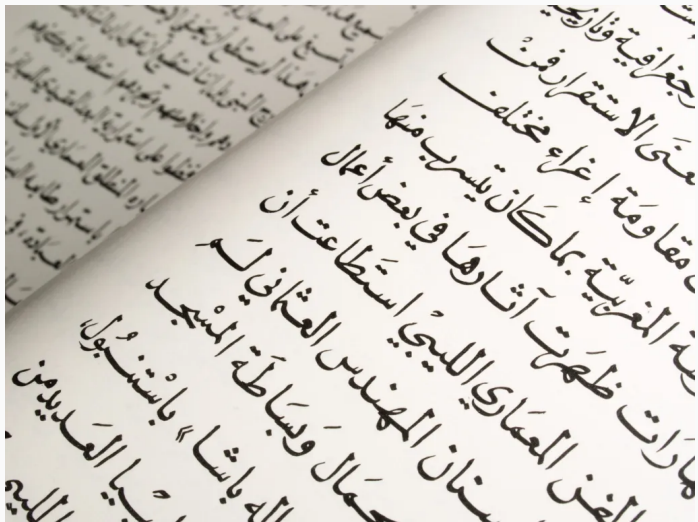
(vertical, hangeul & hanja)

모든 人間은 태어날 때부터
自由로우며, 그 尊嚴과 權利에
있어 同等하다. 人間은
天賦적으로 理性과 良心을
賦與받았으며, 서로 兄弟愛의
精神으로 行動하여야 한다

Sample text

(vertical, hangeul only)

모든 인간은 태어날 때부터
자유로우며 그 존엄과 권리에
있어 동등하다. 인간은
천부적으로 이성과 양심을
부여받았으며 서로 형제애의
정신으로 행동하여야 한다.



- Абстрактный текст

Этапы рендеринга текста

- Абстрактный текст
- + *кодировка* \implies машинное представление текста

Этапы рендеринга текста

- Абстрактный текст
- + *кодировка* \implies машинное представление текста
- + *шрифт* + настройки *шейпинга* (*shaping*) \implies набор глифов (изображений символов) и их координат

Этапы рендеринга текста

- Абстрактный текст
- + *кодировка* \implies машинное представление текста
- + *шрифт* + настройки *шейпинга* (*shaping*) \implies набор глифов (изображений символов) и их координат
- + *алгоритм* рендеринга \implies нарисованный текст

- Описывают машинное представление текста, т.е. соответствие **последовательностей символов** и **последовательностей бит**

- 7 бит (обычно дополняется нулевым старшим битом до 8 бит), первые 32 символа - управляющие (`\r`, `\n`, `\t`, ...), остальные 96 – буквы английского алфавита (большие и маленькие) и прочие символы (различные скобки, арифметические операции, пунктуация, пробел, ...)

Кодировки: ASCII

- 7 бит (обычно дополняется нулевым старшим битом до 8 бит), первые 32 символа - управляющие (`\r`, `\n`, `\t`, ...), остальные 96 – буквы английского алфавита (большие и маленькие) и прочие символы (различные скобки, арифметические операции, пунктуация, пробел, ...)
- Многие кодировки совпадают с ASCII в диапазоне 0-127 или 32-127

- **ISO/IEC 8859** – 15 разных вариантов (ISO/IEC 8859-5 для русского языка)

- **ISO/IEC 8859** – 15 разных вариантов (ISO/IEC 8859-5 для русского языка)
- **Code page XXX** – много разных кодировок для DOS (Code page 866 для русского языка)

- **ISO/IEC 8859** – 15 разных вариантов (ISO/IEC 8859-5 для русского языка)
- **Code page XXX** – много разных кодировок для DOS (Code page 866 для русского языка)
- **Windows code pages** (Windows-1251 для русского языка)

- **ISO/IEC 8859** – 15 разных вариантов (ISO/IEC 8859-5 для русского языка)
- **Code page XXX** – много разных кодировок для DOS (Code page 866 для русского языка)
- **Windows code pages** (Windows-1251 для русского языка)
- **KOI-8** и вариации – для русского языка

Кодировки: 8-битные

- **ISO/IEC 8859** – 15 разных вариантов (ISO/IEC 8859-5 для русского языка)
- **Code page XXX** – много разных кодировок для DOS (Code page 866 для русского языка)
- **Windows code pages** (Windows-1251 для русского языка)
- **KOI-8** и вариации – для русского языка
- И т.д.

- **Unicode** – стандарт, описывающий соответствие абстрактных символов целочисленным кодам (*code points*) в диапазоне **0..10FFFFh** исключая **D800h..DFFFh** для суррогатных пар в UTF-16 (итого 1112064 code point'a), и рекомендации по их интерпретации и визуализации

- **Unicode** – стандарт, описывающий соответствие абстрактных символов целочисленным кодам (*code points*) в диапазоне **0..10FFFFh** исключая **D800h..DFFFh** для суррогатных пар в UTF-16 (итого 1112064 code point'a), и рекомендации по их интерпретации и визуализации
- На сегодняшний день описывает 149186 символов (в прошлом году было 144697)

- **Unicode** – стандарт, описывающий соответствие абстрактных символов целочисленным кодам (*code points*) в диапазоне **0..10FFFFh** исключая **D800h..DFFFh** для суррогатных пар в UTF-16 (итого 1112064 code point'a), и рекомендации по их интерпретации и визуализации
- На сегодняшний день описывает 149186 символов (в прошлом году было 144697)
- Сам unicode – **не кодировка**, но есть основанные на нём кодировки

- **UTF-8**: от 1 до 4 байт на символ, совпадает с ASCII в диапазоне `0..7Fh`, самая распространённая сегодня кодировка (95% интернета)

- **UTF-8**: от 1 до 4 байт на символ, совпадает с ASCII в диапазоне `0..7Fh`, самая распространённая сегодня кодировка (95% интернета)
- **UCS-2**: устаревшая, 2 байта на символ, не поддерживает весь unicode

- **UTF-8**: от 1 до 4 байт на символ, совпадает с ASCII в диапазоне `0..7Fh`, самая распространённая сегодня кодировка (95% интернета)
- **UCS-2**: устаревшая, 2 байта на символ, не поддерживает весь unicode
- **UTF-16**: 2 или 4 байта на символ

- **UTF-8**: от 1 до 4 байт на символ, совпадает с ASCII в диапазоне **0 . . 7Fh**, самая распространённая сегодня кодировка (95% интернета)
- **UCS-2**: устаревшая, 2 байта на символ, не поддерживает весь unicode
- **UTF-16**: 2 или 4 байта на символ
- **UTF-32**: 4 байта на символ

Кодировки: unicode

- **UTF-8**: от 1 до 4 байт на символ, совпадает с ASCII в диапазоне **0 . . 7Fh**, самая распространённая сегодня кодировка (95% интернета)
- **UCS-2**: устаревшая, 2 байта на символ, не поддерживает весь unicode
- **UTF-16**: 2 или 4 байта на символ
- **UTF-32**: 4 байта на символ
- **GB 18030**: специальная кодировка для китайских иероглифов (но тоже поддерживает весь unicode)

- **Code point** – один unicode элемент (абстрактный символ)

Code points, глифы и графемы

- **Code point** – один unicode элемент (абстрактный символ)
- **Глиф** – одно изображение в шрифте

Code points, глифы и графемы

- **Code point** – один unicode элемент (абстрактный символ)
- **Глиф** – одно изображение в шрифте
- **Графема** – один визуальный символ (один или несколько глифов)

Code points, глифы и графемы

- **Code point** – один unicode элемент (абстрактный символ)
- **Глиф** – одно изображение в шрифте
- **Графема** – один визуальный символ (один или несколько глифов)
- В общем случае один code point **не соответствует** одному глифу или одной графеме

Code points, глифы и графемы

- **Code point** – один unicode элемент (абстрактный символ)
- **Глиф** – одно изображение в шрифте
- **Графема** – один визуальный символ (один или несколько глифов)
- В общем случае один code point **не соответствует** одному глифу или одной графеме
- Примеры:
 - Два символа **ff** могут быть представлены двумя глифами или одним глифом (лигатурой) **ff**

Code points, глифы и графемы

- **Code point** – один unicode элемент (абстрактный символ)
- **Глиф** – одно изображение в шрифте
- **Графема** – один визуальный символ (один или несколько глифов)
- В общем случае один code point **не соответствует** одному глифу или одной графеме
- Примеры:
 - Два символа `ff` могут быть представлены двумя глифами или одним глифом (лигатурой) `ff`
 - Символ `Ô` может быть одним или двумя code point'ами и одним или двумя (`O + ``) глифами, но считается одной графемой

- Содержит набор *глифов* (изображений символов в каком-либо виде) и правил их использования

- Содержит набор *глифов* (изображений символов в каком-либо виде) и правил их использования
- Виды шрифтов:

- Содержит набор *глифов* (изображений символов в каком-либо виде) и правил их использования
- Виды шрифтов:
 - **Bitmap-шрифты**: глиф – готовое изображение (bitmap)

- Содержит набор *глифов* (изображений символов в каком-либо виде) и правил их использования
- Виды шрифтов:
 - **Bitmap-шрифты**: глиф – готовое изображение (bitmap)
 - **Векторные шрифты**: глиф описывается как геометрическая фигура

- Содержит набор *глифов* (изображений символов в каком-либо виде) и правил их использования
- Виды шрифтов:
 - **Bitmap-шрифты**: глиф – готовое изображение (bitmap)
 - **Векторные шрифты**: глиф описывается как геометрическая фигура
 - **(M)SDF-шрифты**: глиф описывается с помощью *signed distance field* (SDF)

- Современные форматы шрифтов (`.ttf` – TrueType, `.otf` – OpenType) – векторные, описывают границу глифа как набор отрезков и квадратичных кривых Безье

- Современные форматы шрифтов (`.ttf` – TrueType, `.otf` – OpenType) – векторные, описывают границу глифа как набор отрезков и квадратичных кривых Безье
- Bitmap и SDF шрифты часто строятся по векторным шрифтам

- Современные форматы шрифтов (**.ttf** – TrueType, **.otf** – OpenType) – векторные, описывают границу глифа как набор отрезков и квадратичных кривых Безье
- Bitmap и SDF шрифты часто строятся по векторным шрифтам
- **FreeType** – самая распространённая библиотека для чтения векторных шрифтов; умеет растеризовать в bitmap и (с версии 2.11.0, июль 2021) в SDF

Шейпинг (shaping)

- Процесс преобразования последовательности символов в набор отпозиционированных глифов

Шейпинг (shaping)

- Процесс преобразования последовательности символов в набор отпозиционированных глифов
- Может включать в себя:

Шейпинг (shaping)

- Процесс преобразования последовательности символов в набор отпозиционированных глифов
- Может включать в себя:
 - Настройки: направление (слева-направо, справа-налево, сверху-вниз, снизу-вверх), размер шрифта, межбуквенное расстояние, стиль (жирный, курсив, и т.п.)

Шейпинг (shaping)

- Процесс преобразования последовательности символов в набор отпозиционированных глифов
- Может включать в себя:
 - Настройки: направление (слева-направо, справа-налево, сверху-вниз, снизу-вверх), размер шрифта, межбуквенное расстояние, стиль (жирный, курсив, и т.п.)
 - Hinting: сдвиг глифов, чтобы они были лучше выровнены по пиксельной сетке

Шейпинг (shaping)

- Процесс преобразования последовательности символов в набор отпозиционированных глифов
- Может включать в себя:
 - Настройки: направление (слева-направо, справа-налево, сверху-вниз, снизу-вверх), размер шрифта, межбуквенное расстояние, стиль (жирный, курсив, и т.п.)
 - Hinting: сдвиг глифов, чтобы они были лучше выровнены по пиксельной сетке
 - Kerning: изменение расстояния между соседними глифами для лучшего восприятия

Шейпинг (shaping)

- Процесс преобразования последовательности символов в набор отпозиционированных глифов
- Может включать в себя:
 - Настройки: направление (слева-направо, справа-налево, сверху-вниз, снизу-вверх), размер шрифта, межбуквенное расстояние, стиль (жирный, курсив, и т.п.)
 - Hinting: сдвиг глифов, чтобы они были лучше выровнены по пиксельной сетке
 - Kerning: изменение расстояния между соседними глифами для лучшего восприятия
 - Лигатуры: последовательности несвязанных символов, представленные одним глифом (ff, fi, <=>)

Шейпинг (shaping)

- Шейпинг **не занимается** расстановкой строк и абзацев и текстом, а также двунаправленным (bidirectional) текстом (идущим как слева-направо, так и справа-налево)

Шейпинг (shaping)

- Шейпинг **не занимается** расстановкой строк и абзацев и текстом, а также двунаправленным (bidirectional) текстом (идущим как слева-направо, так и справа-налево)
- Для простых моноширинных шрифтов шейпинг может сводиться к расположению глифов на равных расстояниях друг от друга

Шейпинг (shaping)

- Шейпинг **не занимается** расстановкой строк и абзацев и текстом, а также двунаправленным (bidirectional) текстом (идущим как слева-направо, так и справа-налево)
- Для простых моноширинных шрифтов шейпинг может сводиться к расположению глифов на равных расстояниях друг от друга
- **harfbuzz** – одна из самых распространённых библиотек для шейпинга текста, используется всеми на свете

Шейпинг (shaping)

- Шейпинг **не занимается** расстановкой строк и абзацев и текстом, а также двунаправленным (bidirectional) текстом (идущим как слева-направо, так и справа-налево)
- Для простых моноширинных шрифтов шейпинг может сводиться к расположению глифов на равных расстояниях друг от друга
- **harfbuzz** – одна из самых распространённых библиотек для шейпинга текста, используется всеми на свете
- **FreeType** позволяет сделать шейпинг, но хуже, чем harfbuzz

Шейпинг (shaping)

- Шейпинг **не занимается** расстановкой строк и абзацев и текстом, а также двунаправленным (bidirectional) текстом (идущим как слева-направо, так и справа-налево)
- Для простых моноширинных шрифтов шейпинг может сводиться к расположению глифов на равных расстояниях друг от друга
- **harfbuzz** – одна из самых распространённых библиотек для шейпинга текста, используется всеми на свете
- **FreeType** позволяет сделать шейпинг, но хуже, чем harfbuzz
- **FriBidi** – библиотека для двунаправленного текста

abcfgop AO *abcfgop*
abcfgop AO *abcfgop*

維基百科
維基百科國際
維基百科
維基百科國際

abcfgop

abcfgop

Kerning

AV Wa
No kerning



AV Wa
Kerning applied



$AE \rightarrow \text{Æ}$	$ij \rightarrow \text{ij}$
$ae \rightarrow \text{æ}$	$st \rightarrow \text{ſt}$
$OE \rightarrow \text{Œ}$	$ft \rightarrow \text{ft}$
$oe \rightarrow \text{œ}$	$et \rightarrow \text{\&}$
$ff \rightarrow \text{ff}$	$fs \rightarrow \text{ß}$
$fi \rightarrow \text{fi}$	$ffi \rightarrow \text{ffi}$

Рендеринг bitmap-шрифтов

- Обычно представлены в виде texture atlas: одна текстура, содержащая все глифы шрифта

Рендеринг bitmap-шрифтов

- Обычно представлены в виде texture atlas: одна текстура, содержащая все глифы шрифта
- Содержит информацию о расположении глифов в текстуре (текстурные координаты левого верхнего и правого нижнего пикселя)

Рендеринг bitmap-шрифтов

- Обычно представлены в виде texture atlas: одна текстура, содержащая все глифы шрифта
- Содержит информацию о расположении глифов в текстуре (текстурные координаты левого верхнего и правого нижнего пикселя)
- Глифы рисуются как текстурированные прямоугольники

Рендеринг bitmap-шрифтов

- Обычно представлены в виде texture atlas: одна текстура, содержащая все глифы шрифта
- Содержит информацию о расположении глифов в текстуре (текстурные координаты левого верхнего и правого нижнего пикселя)
- Глифы рисуются как текстурированные прямоугольники
- Плохо ведёт себя при масштабировании (как увеличении, так и уменьшении), bitmap'ы не особо помогают

Рендеринг bitmap-шрифтов

- Обычно представлены в виде texture atlas: одна текстура, содержащая все глифы шрифта
- Содержит информацию о расположении глифов в текстуре (текстурные координаты левого верхнего и правого нижнего пикселя)
- Глифы рисуются как текстурированные прямоугольники
- Плохо ведёт себя при масштабировании (как увеличении, так и уменьшении), bitmap'ы не особо помогают
- Очень прост в реализации

Рендеринг bitmap-шрифтов

- Обычно представлены в виде texture atlas: одна текстура, содержащая все глифы шрифта
- Содержит информацию о расположении глифов в текстуре (текстурные координаты левого верхнего и правого нижнего пикселя)
- Глифы рисуются как текстурированные прямоугольники
- Плохо ведёт себя при масштабировании (как увеличении, так и уменьшении), bitmap'ы не особо помогают
- Очень прост в реализации
- Часто используется для дебажного текста, инди-игр, и т.п.

! " # \$ % & ' () * + , - . / 0 1
2 3 4 5 6 7 8 9 : ; < = > ? @ A B C
D E F G H I J K L M N O P Q R S T U
V W X Y Z [\] ^ _ ` a b c d e f g
h i j k l m n o p q r s t u v w x y
z { | } ~

Bitmap-шрифт: описание в коде

```
struct bitmap_font
{
    GLuint texture_id;

    struct glyph
    {
        vec2 top_left;
        vec2 bottom_right;
    };

    std::unordered_map<std::char32_t, glyph> glyphs;
};
```

Bitmap-шрифт: фрагментный шейдер

```
uniform sampler2D font_texture;
uniform vec3 text_color;

in vec2 texcoord;

layout (location = 0) out vec4 out_color;

void main()
{
    float alpha = texture(font_texture, texcoord).r;
    out_color = vec4(text_color, alpha);
}
```

Рендеринг векторных шрифтов

- Глиф описывается как набор геометрических фигур (фигура может описывать 'дырку' в другой фигуре, как дырка в букве 'O'), граница фигуры – набор отрезков и квадратичных кривых Безье

Рендеринг векторных шрифтов

- Глиф описывается как набор геометрических фигур (фигура может описывать 'дырку' в другой фигуре, как дырка в букве 'O'), граница фигуры – набор отрезков и квадратичных кривых Безье
- Много разных способов рендеринга:

Рендеринг векторных шрифтов

- Глиф описывается как набор геометрических фигур (фигура может описывать 'дырку' в другой фигуре, как дырка в букве 'O'), граница фигуры – набор отрезков и квадратичных кривых Безье
- Много разных способов рендеринга:
 - Аппроксимация набором треугольников

Рендеринг векторных шрифтов

- Глиф описывается как набор геометрических фигур (фигура может описывать 'дырку' в другой фигуре, как дырка в букве 'О'), граница фигуры – набор отрезков и квадратичных кривых Безье
- Много разных способов рендеринга:
 - Аппроксимация набором треугольников
 - Упаковка фигур в текстуру + шейдер, который честно вычисляет площадь пересечения фигуры и пикселя

Рендеринг векторных шрифтов

- Глиф описывается как набор геометрических фигур (фигура может описывать 'дырку' в другой фигуре, как дырка в букве 'O'), граница фигуры – набор отрезков и квадратичных кривых Безье
- Много разных способов рендеринга:
 - Аппроксимация набором треугольников
 - Упаковка фигур в текстуру + шейдер, который честно вычисляет площадь пересечения фигуры и пикселя
 - Полигональная аппроксимация глифа (рисуеться с использованием stencil буфера) + треугольник со специальным шейдером для каждой кривой Безье

Рендеринг векторных шрифтов

- Глиф описывается как набор геометрических фигур (фигура может описывать 'дырку' в другой фигуре, как дырка в букве 'O'), граница фигуры – набор отрезков и квадратичных кривых Безье
- Много разных способов рендеринга:
 - Аппроксимация набором треугольников
 - Упаковка фигур в текстуру + шейдер, который честно вычисляет площадь пересечения фигуры и пикселя
 - Полигональная аппроксимация глифа (рисуеться с использованием stencil буфера) + треугольник со специальным шейдером для каждой кривой Безье
 - Slug algorithm (запатентован)

Рендеринг векторных шрифтов

- Глиф описывается как набор геометрических фигур (фигура может описывать 'дырку' в другой фигуре, как дырка в букве 'O'), граница фигуры – набор отрезков и квадратичных кривых Безье
- Много разных способов рендеринга:
 - Аппроксимация набором треугольников
 - Запакровка фигур в текстуру + шейдер, который честно вычисляет площадь пересечения фигуры и пикселя
 - Полигональная аппроксимация глифа (рисуеться с использованием stencil буфера) + треугольник со специальным шейдером для каждой кривой Безье
 - Slug algorithm (запатентован)
- Обычно легко переносит масштабирование

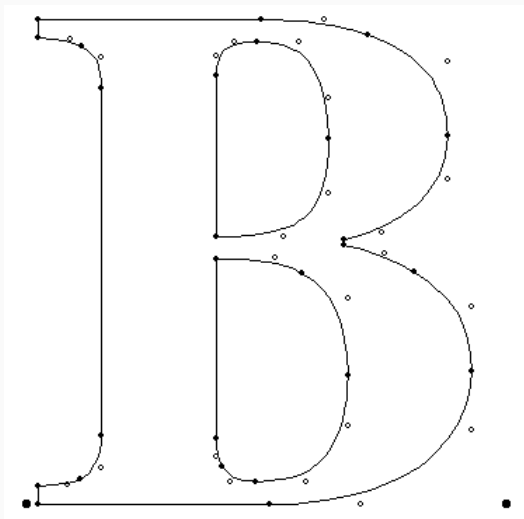
Рендеринг векторных шрифтов

- Глиф описывается как набор геометрических фигур (фигура может описывать 'дырку' в другой фигуре, как дырка в букве 'O'), граница фигуры – набор отрезков и квадратичных кривых Безье
- Много разных способов рендеринга:
 - Аппроксимация набором треугольников
 - Запакровка фигур в текстуру + шейдер, который честно вычисляет площадь пересечения фигуры и пикселя
 - Полигональная аппроксимация глифа (рисуеться с использованием stencil буфера) + треугольник со специальным шейдером для каждой кривой Безье
 - Slug algorithm (запатентован)
- Обычно легко переносит масштабирование
- Сложен в реализации

Рендеринг векторных шрифтов

- Глиф описывается как набор геометрических фигур (фигура может описывать 'дырку' в другой фигуре, как дырка в букве 'O'), граница фигуры – набор отрезков и квадратичных кривых Безье
- Много разных способов рендеринга:
 - Аппроксимация набором треугольников
 - Запакровка фигур в текстуру + шейдер, который честно вычисляет площадь пересечения фигуры и пикселя
 - Полигональная аппроксимация глифа (рисуеться с использованием stencil буфера) + треугольник со специальным шейдером для каждой кривой Безье
 - Slug algorithm (запатентован)
- Обычно легко переносит масштабирование
- Сложен в реализации
- Используется для текста максимально возможного качества

Векторный глиф



Slug algorithm



Signed distance field (SDF)

- Описание двумерного или трёхмерного объекта/фигуры *функцией расстояния* до границы объекта

Signed distance field (SDF)

- Описание двумерного или трёхмерного объекта/фигуры *функцией расстояния* до границы объекта
- Обычно положительна снаружи объекта и отрицательна внутри (поэтому *signed*), $f(p) = 0$ – граница объекта

Signed distance field (SDF)

- Описание двумерного или трёхмерного объекта/фигуры *функцией расстояния* до границы объекта
- Обычно положительна снаружи объекта и отрицательна внутри (поэтому *signed*), $f(p) = 0$ – граница объекта
- SDF может быть представлена явной формулой (напр. $f(p) = \|p - O\| - R$ – расстояние до сферы радиуса R с центром в точке O) или текстурой

Signed distance field (SDF)

- Описание двумерного или трёхмерного объекта/фигуры *функцией расстояния* до границы объекта
- Обычно положительна снаружи объекта и отрицательна внутри (поэтому *signed*), $f(p) = 0$ – граница объекта
- SDF может быть представлена явной формулой (напр. $f(p) = \|p - O\| - R$ – расстояние до сферы радиуса R с центром в точке O) или текстурой
- SDF-сцены часто используются для экспериментального рендеринга и удобны для raymarching'a

Рендеринг SDF-шрифтов (Valve, 2007)

- Описывается так же, как bitmap-шрифт, но текстура хранит значения SDF для глифов

Рендеринг SDF-шрифтов (Valve, 2007)

- Описывается так же, как bitmap-шрифт, но текстура хранит значения SDF для глифов
- Фрагментный шейдер читает значение SDF из текстуры шрифта: если оно меньше 0, то пиксель находится внутри глифа (e.g. чёрный пиксель), иначе – нет (e.g. прозрачный пиксель)

Рендеринг SDF-шрифтов (Valve, 2007)

- Описывается так же, как bitmap-шрифт, но текстура хранит значения SDF для глифов
- Фрагментный шейдер читает значение SDF из текстуры шрифта: если оно меньше 0, то пиксель находится внутри глифа (e.g. чёрный пиксель), иначе – нет (e.g. прозрачный пиксель)
- Прост в реализации

Рендеринг SDF-шрифтов (Valve, 2007)

- Описывается так же, как bitmap-шрифт, но текстура хранит значения SDF для глифов
- Фрагментный шейдер читает значение SDF из текстуры шрифта: если оно меньше 0, то пиксель находится внутри глифа (e.g. чёрный пиксель), иначе – нет (e.g. прозрачный пиксель)
- Прост в реализации
- Требуется чуть больше места под глифы, но менее требователен к разрешению

Рендеринг SDF-шрифтов (Valve, 2007)

- Описывается так же, как bitmap-шрифт, но текстура хранит значения SDF для глифов
- Фрагментный шейдер читает значение SDF из текстуры шрифта: если оно меньше 0, то пиксель находится внутри глифа (e.g. чёрный пиксель), иначе – нет (e.g. прозрачный пиксель)
- Прост в реализации
- Требуется чуть больше места под глифы, но менее требователен к разрешению
- Неплохо масштабируется (есть артефакты, но менее серьёзные, чем для bitmap-шрифтов)

Рендеринг SDF-шрифтов (Valve, 2007)

- Описывается так же, как bitmap-шрифт, но текстура хранит значения SDF для глифов
- Фрагментный шейдер читает значение SDF из текстуры шрифта: если оно меньше 0, то пиксель находится внутри глифа (e.g. чёрный пиксель), иначе – нет (e.g. прозрачный пиксель)
- Прост в реализации
- Требуется чуть больше места под глифы, но менее требователен к разрешению
- Неплохо масштабируется (есть артефакты, но менее серьёзные, чем для bitmap-шрифтов)
- Один из самых популярных способов рендеринга шрифтов

@}{()jll[\$Q%OGC&S#9/\

U389Y06qb?pdJfWMA YV

XRDKTNHZPBE4F25Lkh1

!lll7t;oaecsmnurwxvz:><

+^*=|||~\.-



- Тестуры позволяют хранить значения от 0 до 1

- Тестуры позволяют хранить значения от 0 до 1
- Мы хотим хранить произвольные, но не очень большие по модулю числа (расстояние в пикселях)

- Тестуры позволяют хранить значения от 0 до 1
- Мы хотим хранить произвольные, но не очень большие по модулю числа (расстояние в пикселях)
- \implies В текстуре придётся хранить что-то в духе $0.5 + sdf / scale$, где $scale$ – максимальное представимое расстояние

- Тестуры позволяют хранить значения от 0 до 1
- Мы хотим хранить произвольные, но не очень большие по модулю числа (расстояние в пикселях)
- \implies В текстуре придётся хранить что-то в духе $0.5 + sdf / scale$, где $scale$ – максимальное представимое расстояние
- \implies При чтении из текстуры нужно применять обратное преобразование $(sdf - 0.5) \cdot scale$

- Тестуры позволяют хранить значения от 0 до 1
- Мы хотим хранить произвольные, но не очень большие по модулю числа (расстояние в пикселях)
- \implies В текстуре придётся хранить что-то в духе $0.5 + sdf / scale$, где $scale$ – максимальное представимое расстояние
- \implies При чтении из текстуры нужно применять обратное преобразование $(sdf - 0.5) \cdot scale$
- Лучше включить для этой текстуры анизотропную фильтрацию, чтобы текст хорошо выглядел ‘сбоку’

- Можно легко реализовать много дополнительных эффектов:

- Можно легко реализовать много дополнительных эффектов:
 - Сглаживание: вместо жёсткой границы плавно меняем прозрачность пикселя в зависимости от значения SDF

- Можно легко реализовать много дополнительных эффектов:
 - Сглаживание: вместо жёсткой границы плавно меняем прозрачность пикселя в зависимости от значения SDF
 - Сглаживание с учётом масштаба и перспективы: по dF_x , dF_y можно вычислить диапазон значений SDF, чтобы сглаживание было ровно в 1 пиксель в экранных координатах

- Можно легко реализовать много дополнительных эффектов:
 - Сглаживание: вместо жёсткой границы плавно меняем прозрачность пикселя в зависимости от значения SDF
 - Сглаживание с учётом масштаба и перспективы: по dF_{fx} , dF_{fy} можно вычислить диапазон значений SDF, чтобы сглаживание было ровно в 1 пиксель в экранных координатах
 - Обводка текста другим цветом: рисуем цвет обводки, если $0 \leq f(p) \leq \epsilon$

- Можно легко реализовать много дополнительных эффектов:
 - Сглаживание: вместо жёсткой границы плавно меняем прозрачность пикселя в зависимости от значения SDF
 - Сглаживание с учётом масштаба и перспективы: по $dFfx$, $dFfy$ можно вычислить диапазон значений SDF, чтобы сглаживание было ровно в 1 пиксель в экранных координатах
 - Обводка текста другим цветом: рисуем цвет обводки, если $0 \leq f(p) \leq \epsilon$
 - Псевдотрёхмерный текст: по градиенту SDF можно восстановить нормаль к глифу

Text

SDF-шрифт: фрагментный шейдер

```
uniform sampler2D sdfTexture;
uniform float sdfScale;
uniform vec3 textColor;

in vec2 texcoord;

layout (location = 0) out vec4 out_color;

void main()
{
    float textureValue = texture(sdfTexture, texcoord).r;
    float sdfValue = sdfScale * (textureValue - 0.5);
    // сглаживание
    float alpha = smoothstep(-0.5, 0.5, sdfValue);
    out_color = vec4(textColor, alpha);
}
```

- У SDF-текста есть типичные артефакты: острые углы сглаживаются, из-за чего приходится брать SDF-текстуру большого разрешения

- У SDF-текста есть типичные артефакты: острые углы сглаживаются, из-за чего приходится брать SDF-текстуру большого разрешения
- Идея: билинейная интерполяция не портит прямые линии \implies представим глиф *пересечением* нескольких объектов, чтобы острые углы, представленные пересечением нескольких прямых, не сглаживались

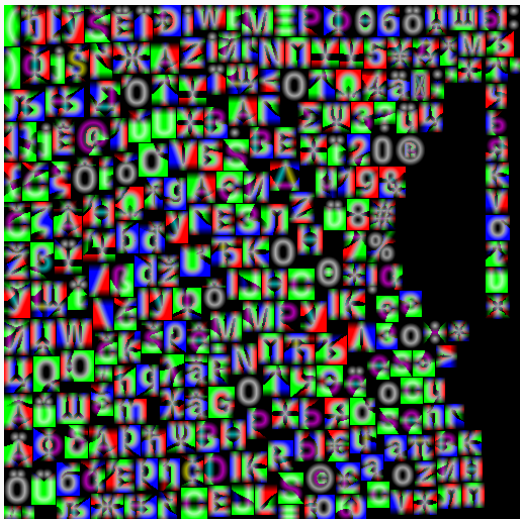
- У SDF-текста есть типичные артефакты: острые углы сглаживаются, из-за чего приходится брать SDF-текстуру большого разрешения
- Идея: билинейная интерполяция не портит прямые линии \implies представим глиф *пересечением* нескольких объектов, чтобы острые углы, представленные пересечением нескольких прямых, не сглаживались
- В текстуре есть 4 канала (RGBA) \implies можем сохранить сразу 4 разных SDF в одной текстуре!

- **MSDF:** Multi-channel signed distance field

- **MSDF**: Multi-channel signed distance field
- На практике оказывается, что хватает 3 каналов, т.е. трёх различных SDF

- **MSDF**: Multi-channel signed distance field
- На практике оказывается, что хватает 3 каналов, т.е. трёх различных SDF
- Вместо пересечения (т.е. минимума из трёх значений SDF) лучше брать *медиану* (т.е. значение посередине между двумя другими)

- **MSDF**: Multi-channel signed distance field
- На практике оказывается, что хватает 3 каналов, т.е. трёх различных SDF
- Вместо пересечения (т.е. минимума из трёх значений SDF) лучше брать *медиану* (т.е. значение посередине между двумя другими)
- Есть инструменты (программа, библиотека, сайт) для генерации таких текстур по шрифту



MSDF: сравнение с SDF



16×16



16×16



32×32



MSDF: пример кода

```
uniform sampler2D sdfTexture;
uniform float sdfScale;
uniform vec3 textColor;

in vec2 texcoord;

layout (location = 0) out vec4 out_color;

float median(vec3 v) {
    return max(min(v.r, v.g), min(max(v.r, v.g), v.b));
}

void main()
{
    float textureValue =
        median(texture(sdfTexture, texcoord).rgb);
    float sdfValue = sdfScale * (textureValue - 0.5);
    float alpha = smoothstep(-0.5, 0.5, sdfValue);
    out_color = vec4(textColor, alpha);
}
```

- FreeType
- harfbuzz
- FriBidi

- `Тutorial по рендерингу bitmap-шрифтов`
- `Один способ рендеринга векторных шрифтов`
- `Другой способ рендеринга векторных шрифтов`
- `Slug algorithm`
- `Slug library`

- Статья от Valve про SDF-текст
- Тьюториал по рендерингу SDF-шрифтов
- SDF font generator
- Репозиторий с генератором MSDF-шрифтов от автора этой техники
- Shape Decomposition for Multi-channel Distance Fields (Chlumský, 2015)
- MSDF font generator

Чем заняться дальше?

- Очень много источников света
- Очень много объектов
- Очень большая сцена
- Сложные объекты
- Другие API
- Raytracing
- Статьи и конференции

- Deferred shading
- Tiled/Clustered shading
- Compute shaders (OpenGL 4.3 или расширение)
- **Площадные источники света**

- Batching
- Frustum culling
- Occlusion culling
- LOD

Очень большая сцена (e.g. планета)

- LOD (e.g. для ландшафта)
- Проблемы с точностью (`float` не хватает; нужно рисовать относительно некой anchor-точки в `double`)
- Проблемы с буфером глубины (`reversed z`)

- Вода: отражение + преломление (по Френелю) + 'туман' в плотности воды + **волны Герстнера**
- Растительность (vegetation): трава, кусты, деревья
- Облака + небо (volume rendering)
- Сложные BRDF

- OpenGL 4.0+: compute shaders, indirect rendering, direct state access
- Vulkan: vulkan-tutorial.com
- WebGPU: [WebGPU C++ Guide](#)

- Ray Tracing in One Weekend (He real-time)
- Vulkan raytracing tutorial
- Voxel cone tracing
- Surflet-based GI
- ReSTIR GI

- GPU Gems 1, 2, 3
- SIGGRAPH (e.g. 2022)
- <https://www.gdcvault.com/free/>
- Graphics Programming Weekly

- Область real-time рендеринга очень активно развивается
- О рисовании любого объекта/эффекта можно найти десятки статей и даже PhD
- Есть тысячи туториалов по всему на свете
- Не бойтесь гуглить и читать непонятные статьи, со временем станет понятнее
- Не бойтесь писать мне :)