

Компьютерная графика

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

2025

Рендеринг текста

- Текст – очень сложная штука

Рендеринг текста

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

Рендеринг текста

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

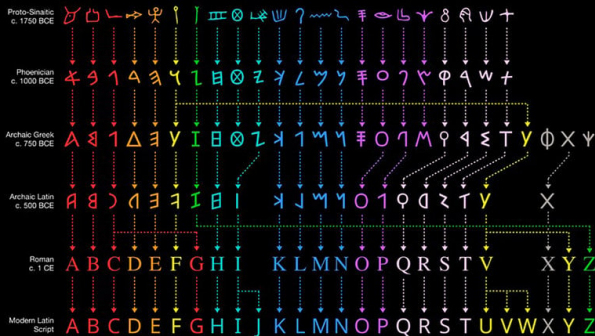
Рендеринг текста

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

Рендеринг текста

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

Sample text

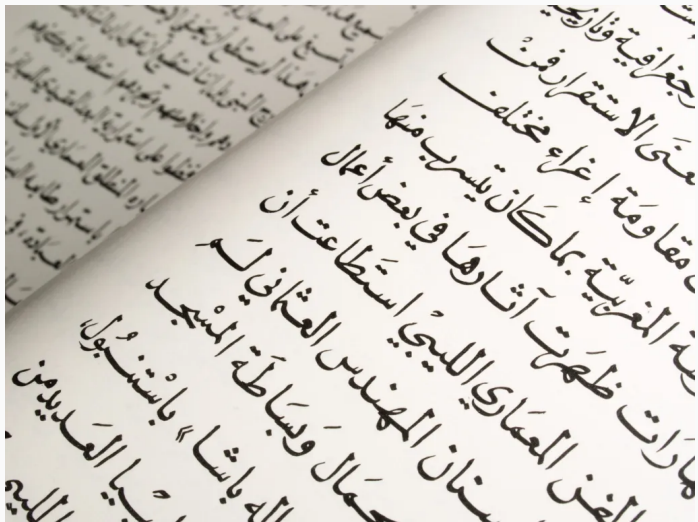
(vertical, hangeul & hanja)

모든 人間은 태어날 때부터
自由로우며, 그 尊嚴과 權利에
있어 同等하다. 人間은
天賦的으로 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

- **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 `0x0..0x10FFFFh` исключая `0xD800h..0xDFFFh` 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 `0x0..0x10FFFFh` исключая `0xD800h..0xDFFFh` для суррогатных пар в UTF-16 (итого 1112064 code point'a), и рекомендации по их интерпретации и визуализации
- На сегодняшний день описывает 154998 символов (в прошлом году было 149186)

Кодировки: unicode

- **Unicode** – стандарт, описывающий соответствие абстрактных символов целочисленным кодам (*code points*) в диапазоне `0x0..0x10FFFFh` исключая `0xD800h..0xDFFFh` для суррогатных пар в UTF-16 (итого 1112064 code point'a), и рекомендации по их интерпретации и визуализации
- На сегодняшний день описывает 154998 символов (в прошлом году было 149186)
- Сам unicode – **не кодировка**, а таблица соответствия, 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

- **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)

- В кодировках UTF-8 и UTF-16 символ занимает *переменное* количество байт – в этой кодировке сложнее двигаться по тексту на целое число code point'ов

- В кодировках UTF-8 и UTF-16 символ занимает *переменное* количество байт – в этой кодировке сложнее двигаться по тексту на целое число code point'ов
- UTF-16 часто используется в WinAPI как альтернатива более старым функциям, поддерживавшим только 8-битные кодировки

- В кодировках UTF-8 и UTF-16 символ занимает *переменное* количество байт – в этой кодировке сложнее двигаться по тексту на целое число code point'ов
- UTF-16 часто используется в WinAPI как альтернатива более старым функциям, поддерживавшим только 8-битные кодировки
- **Моё мнение:** лучше хранить текст в UTF-8, редактировать его в UTF-32, и конвертировать в UTF-16 только для передачи в WinAPI :)

- **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, а также двунаправленным (bidirectional) текстом (идущим как слева-направо, так и справа-налево)

Шейпинг (shaping)

- Шейпинг **не занимается** расстановкой строк и абзацев, а также двунаправленным (bidirectional) текстом (идушим 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, 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, а также двунаправленным (bidirectional) текстом (идушим 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, 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

This image shows the text 'AV' and 'Wa' in a serif font without kerning applied. A vertical blue line is positioned at the right edge of the 'A' in 'AV', and another vertical blue line is at the right edge of the 'W' in 'Wa'. There is a noticeable gap between the 'A' and 'V' in 'AV', and a gap between the 'W' and 'a' in 'Wa'.

AV Wa
Kerning applied

This image shows the text 'AV' and 'Wa' in the same serif font, but with kerning applied. A vertical blue line is positioned at the right edge of the 'A' in 'AV', and another vertical blue line is at the right edge of the 'W' in 'Wa'. The space between the 'A' and 'V' in 'AV' is now filled, and the space between the 'W' and 'a' in 'Wa' is also filled, making the pairs look more cohesive.

$AE \rightarrow \mathcal{A}E$	$ij \rightarrow \dot{i}\dot{j}$
$ae \rightarrow \mathfrak{a}e$	$st \rightarrow \hat{s}t$
$OE \rightarrow \mathcal{O}E$	$ft \rightarrow \mathcal{f}t$
$oe \rightarrow \mathfrak{o}e$	$et \rightarrow \&$
$ff \rightarrow \mathcal{f}\mathcal{f}$	$fs \rightarrow \beta$
$fi \rightarrow \mathcal{f}i$	$ffi \rightarrow \mathcal{f}\mathcal{f}i$

- 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
- Содержат 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
- Содержат 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
- Содержат информацию о расположении глифов в текстуре (текстурные координаты левого верхнего и правого нижнего пикселя)
- Глифы рисуются как текстурированные прямоугольники
- Плохо ведут 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
- Содержат информацию о расположении глифов в текстуре (текстурные координаты левого верхнего и правого нижнего пикселя)
- Глифы рисуются как текстурированные прямоугольники
- Плохо ведут себя при масштабировании (как увеличении, так и уменьшении), bitmap'ы не особо помогают
- Очень просты в реализации

Рендеринг bitmap-шрифтов

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

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
 - Заpackовка 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
 - Заpackовка 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
 - Заpackовка 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

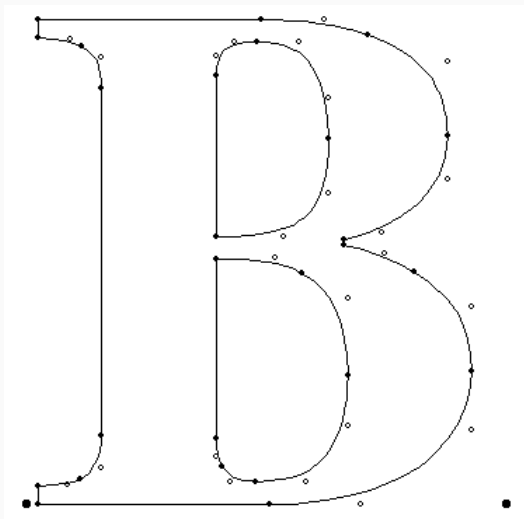
Рендеринг векторных шрифтов

- 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), комбинацией более простых SDF, 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), комбинацией более простых SDF, или текстурой
- SDF-сцены часто используются для экспериментального рендеринга и удобны для raymarching'a

Signed distance field (SDF)

- SDF-сцены часто используются для экспериментального рендеринга и удобны для raymarching'a

Signed distance field (SDF)

- SDF-сцены часто используются для экспериментального рендеринга и удобны для raymarching'a
- Часто, используемые на практике функции не являются SDF, и удовлетворяют более слабым условиям (например, $f(p)$ не больше расстояния до объекта, или $|\nabla f| \leq L$)

Signed distance field (SDF)

- SDF-сцены часто используются для экспериментального рендеринга и удобны для raymarching'a
- Часто, используемые на практике функции не являются SDF, и удовлетворяют более слабым условиям (например, $f(p)$ не больше расстояния до объекта, или $|\nabla f| \leq L$)
- Блог Inigo Quilez (iquilezles.org) – один из лучших ресурсов про SDF и рендеринг с их помощью

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/\n
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 $(sdf - 0.5) \cdot scale$
- Для трёхмерного текста 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!

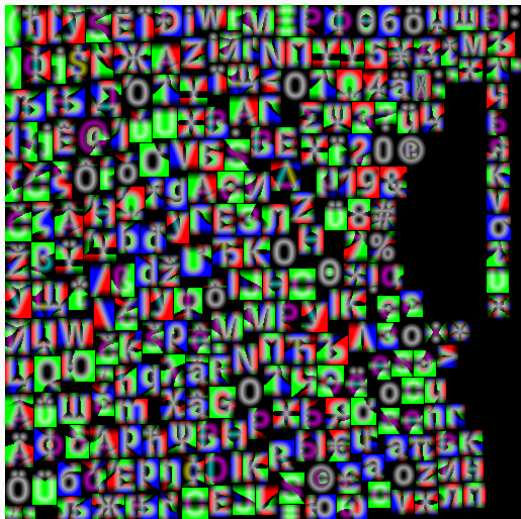
- **MSDF**: Multi-channel signed distance field

- **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)
- Эффекты на этих трёх каналах уже не реализовать, но для этого можно записать обычную SDF в четвёртый (альфа) канал

- **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)
- Эффекты на этих трёх каналах уже не реализовать, но для этого можно записать обычную SDF в четвёртый (альфа) канал
- 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)
- **Линейные и площадные источники света** с помощью linearly transformed cosines (LTC)

- Batching
- Frustum culling
- Occlusion culling
- LOD

Очень большая сцена (e.g. планета)

- Иерархический LOD для ландшафта (например, в виде квадродерева)
- 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' или честное рассеивание в плотности воды + **волны Герстнера**
- Растительность (vegetation): трава, кусты, деревья (см. **статья в GPU Gems, доклад про траву в Ghost of Tsushima, доклад про vegetation в Horizon Zero Dawn**)
- Облака + небо (volume rendering) (см. **доклад про облака в Horizon Zero Dawn**)
- Сложные BRDF (см. **Crash Course in BRDF Implementation**)

- 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
- Radiance cascades

- 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 :)