# Компьютерная графика

## Лекция 14: рендеринг текста, bitmap-шрифты, векторные шрифты, SDF-шрифты, volume rendering, volume slicing, raymarching

2021

# Рендеринг текста

- Абстрактный текст

# Рендеринг текста

- Абстрактный текст
- $+$ кодировка $\Rightarrow$ машинное представление текста

# Рендеринг текста

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

# Рендеринг текста

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

# Кодировки

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

# Кодировки

- Описывают машинное представление текста, т.е. соответствие последовательностей символов и последовательностей бит
- ASCII: 7 бит (обычно дополняется нулевым старшим битом до 8 бит), первые 32 символа - управляющие (`\r`, `\n`, tab, ...), остальные 96 - буквы английского алфавита (большие и маленькие) и прочие символы (различные скобки, арифметические операции, пунктуация, пробел, ...)

# Кодировки

- Описывают машинное представление текста, т.е. соответствие последовательностей символов и последовательностей бит
- ASCII: 7 бит (обычно дополняется нулевым старшим битом до 8 бит), первые 32 символа - управляющие (`\r`, `\n`, tab, ...), 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

# Кодировки

- Описывают машинное представление текста, т.е. соответствие последовательностей символов и последовательностей бит
- ASCII: 7 бит (обычно дополняется нулевым старшим битом до 8 бит), первые 32 символа - управляющие (\r, \n, tab, ...), 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
- Огромное количество в основном 8-битных кодировок для разных алфавитов и систем:
  - 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
  - etc.

# Кодировки

- Описывают машинное представление текста, т.е. соответствие последовательностей символов и последовательностей бит
- ASCII: 7 бит (обычно дополняется нулевым старшим битом до 8 бит), первые 32 символа - управляющие (`\r`, `\n`, tab, ...), 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
- Огромное количество в основном 8-битных кодировок для разных алфавитов и систем:
  - 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
  - etc.
- Unicode-кодировки

# Unicode

- Unicode - стандарт, описывающий соответствие символов целочисленным кодам в диапазоне `0..10FFFFh` исключая `D800h..DFFFh` (используется для суррогатных пар в UTF-16; всего 1112064 символов), и рекомендации по их интерпретации и визуализации

# Unicode

- Unicode - стандарт, описывающий соответствие символов целочисленным кодам в диапазоне `0..10FFFFh` исключая `D800h..DFFFh` (используется для суррогатных пар в UTF-16; всего 1112064 символов), и рекомендации по их интерпретации и визуализации
- На сегодняшний день описывает 144697 символа

# Unicode

- Unicode - стандарт, описывающий соответствие символов целочисленным кодам в диапазоне `0..10FFFFh` исключая `D800h..DFFFh` (используется для суррогатных пар в UTF-16; всего 1112064 символов), и рекомендации по их интерпретации и визуализации
- На сегодняшний день описывает 144697 символа
- Unicode-кодировки:

# Unicode

- Unicode - стандарт, описывающий соответствие символов целочисленным кодам в диапазоне `0..10FFFFh` исключая `D800h..DFFFh` (используется для суррогатных пар в UTF-16; всего 1112064 символов), и рекомендации по их интерпретации и визуализации
- На сегодняшний день описывает 144697 символа
- Unicode-кодировки:
  - UTF-8: от 1 до 4 байт на символ, совпадает с ASCII в диапазоне `0..7Fh`, самая распространённая сегодня кодировка (95% интернета)

# Unicode

- Unicode - стандарт, описывающий соответствие символов целочисленным кодам в диапазоне `0..10FFFFh` исключая `D800h..DFFFh` (используется для суррогатных пар в UTF-16; всего 1112064 символов), и рекомендации по их интерпретации и визуализации
- На сегодняшний день описывает 144697 символа
- Unicode-кодировки:
  - 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

# Unicode

- Unicode - стандарт, описывающий соответствие символов целочисленным кодам в диапазоне `0..10FFFFh` исключая `D800h..DFFFh` (используется для суррогатных пар в UTF-16; всего 1112064 символов), и рекомендации по их интерпретации и визуализации
- На сегодняшний день описывает 144697 символа
- Unicode-кодировки:
  - 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

# Unicode

- Unicode - стандарт, описывающий соответствие символов целочисленным кодам в диапазоне `0..10FFFFh` исключая `D800h..DFFFh` (используется для суррогатных пар в UTF-16; всего 1112064 символов), и рекомендации по их интерпретации и визуализации
- На сегодняшний день описывает 144697 символа
- Unicode-кодировки:
  - 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

# Unicode

- Unicode - стандарт, описывающий соответствие символов целочисленным кодам в диапазоне `0..10FFFFh` исключая `D800h..DFFFh` (используется для суррогатных пар в UTF-16; всего 1112064 символов), и рекомендации по их интерпретации и визуализации
- На сегодняшний день описывает 144697 символа
- Unicode-кодировки:
  - 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)

# Unicode

- Unicode - стандарт, описывающий соответствие символов целочисленным кодам в диапазоне `0..10FFFFh` исключая `D800h..DFFFh` (используется для суррогатных пар в UTF-16; всего 1112064 символов), и рекомендации по их интерпретации и визуализации
- На сегодняшний день описывает 144697 символа
- Unicode-кодировки:
  - 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)
- N.B.: один символ unicode **не соответствует** одному видимому символу (*графеме*)

# Шрифты

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

# Шрифты

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

# Шрифты

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

# Шрифты

- 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

# Шрифты

- 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
  - SDF-шрифты: глиф описывается с помощью *signed distance field* (SDF)

# Шрифты

- 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
  - 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 (т.е. 2-ого порядка)

# Шрифты

- 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
  - 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 (т.е. 2-ого порядка)
- Bitmap и SDF шрифты часто строятся по векторным шрифтам

# Шрифты

- 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
  - 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 (т.е. 2-ого порядка)
- 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

- Процесс преобразования последовательности символов в набор отпозиционированных глифов

# Шейпинг

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

# Шейпинг

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

# Шейпинг

- Процесс преобразования последовательности символов в набор отпозиционированных глифов
- Может включать в себя:
  - Настройки шейпинга: направление (слева-направо, справа-налево, сверху-вниз, снизу-вверх), размер шрифта, межбуквенное расстояние, стиль (жирный, курсив, и т.п.)
  - Hinting: применение дополнительных преобразований к векторному глифу в зависимости от разрешения

# Шейпинг

- Процесс преобразования последовательности символов в набор отпозиционированных глифов
- Может включать в себя:
  - Настройки шейпинга: направление (слева-направо, справа-налево, сверху-вниз, снизу-вверх), размер шрифта, межбуквенное расстояние, стиль (жирный, курсив, и т.п.)
  - Hinting: применение дополнительных преобразований к векторному глифу в зависимости от разрешения
  - Kerning: изменение расстояния между соседними глифами

# Шейпинг

- Процесс преобразования последовательности символов в набор отпозиционированных глифов
- Может включать в себя:
  - Настройки шейпинга: направление (слева-направо, справа-налево, сверху-вниз, снизу-вверх), размер шрифта, межбуквенное расстояние, стиль (жирный, курсив, и т.п.)
  - Hinting: применение дополнительных преобразований к векторному глифу в зависимости от разрешения
  - Kerning: изменение расстояния между соседними глифами
  - Лигатуры: последовательность несвязанных символов, представленная одним глифом (ff, fi, <=>)

# Шейпинг

- Процесс преобразования последовательности символов в набор отпозиционированных глифов
- Может включать в себя:
  - Настройки шейпинга: направление (слева-направо, справа-налево, сверху-вниз, снизу-вверх), размер шрифта, межбуквенное расстояние, стиль (жирный, курсив, и т.п.)
  - Hinting: применение дополнительных преобразований к векторному глифу в зависимости от разрешения
  - Kerning: изменение расстояния между соседними глифами
  - Лигатуры: последовательность несвязанных символов, представленная одним глифом (ff, fi, <=>)
- Для простых моноширинных шрифтов шейпинг может сводиться к расположению глифов на равных расстояниях друг от друга

# Шейпинг

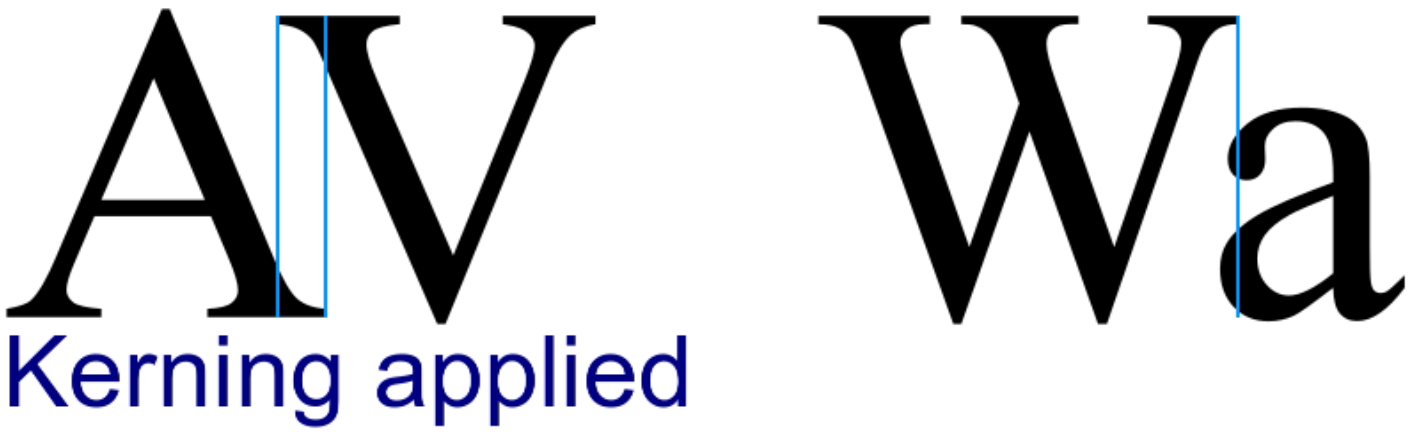
- Процесс преобразования последовательности символов в набор отпозиционированных глифов
- Может включать в себя:
  - Настройки шейпинга: направление (слева-направо, справа-налево, сверху-вниз, снизу-вверх), размер шрифта, межбуквенное расстояние, стиль (жирный, курсив, и т.п.)
  - Hinting: применение дополнительных преобразований к векторному глифу в зависимости от разрешения
  - Kerning: изменение расстояния между соседними глифами
  - Лигатуры: последовательность несвязанных символов, представленная одним глифом (ff, fi, <=>)
- 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
- FreeType позволяет сделать шейпинг, но хуже, чем harfbuzz

# Hinting


abcfgop AO abcfgop
維基百科
維基百科國際
abcfgop AO abcfgop
維基百科
維基百科國際
abcfgop
abcfgop

# Kerning


No kerning




Kerning applied

# Лигатуры

$$AE \to Æ \qquad ij \to ij$$
$$ae \to æ \qquad st \to st$$
$$OE \to Œ \qquad ft \to ft$$
$$oe \to œ \qquad et \to \&$$
$$ff \to ff \qquad fs \to ß$$
$$fi \to fi \qquad ffi \to ffi$$

# 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)
- Плохо ведёт себя при масштабировнии (как увеличении, так и уменьшении), mipmap'ы не особо помогают

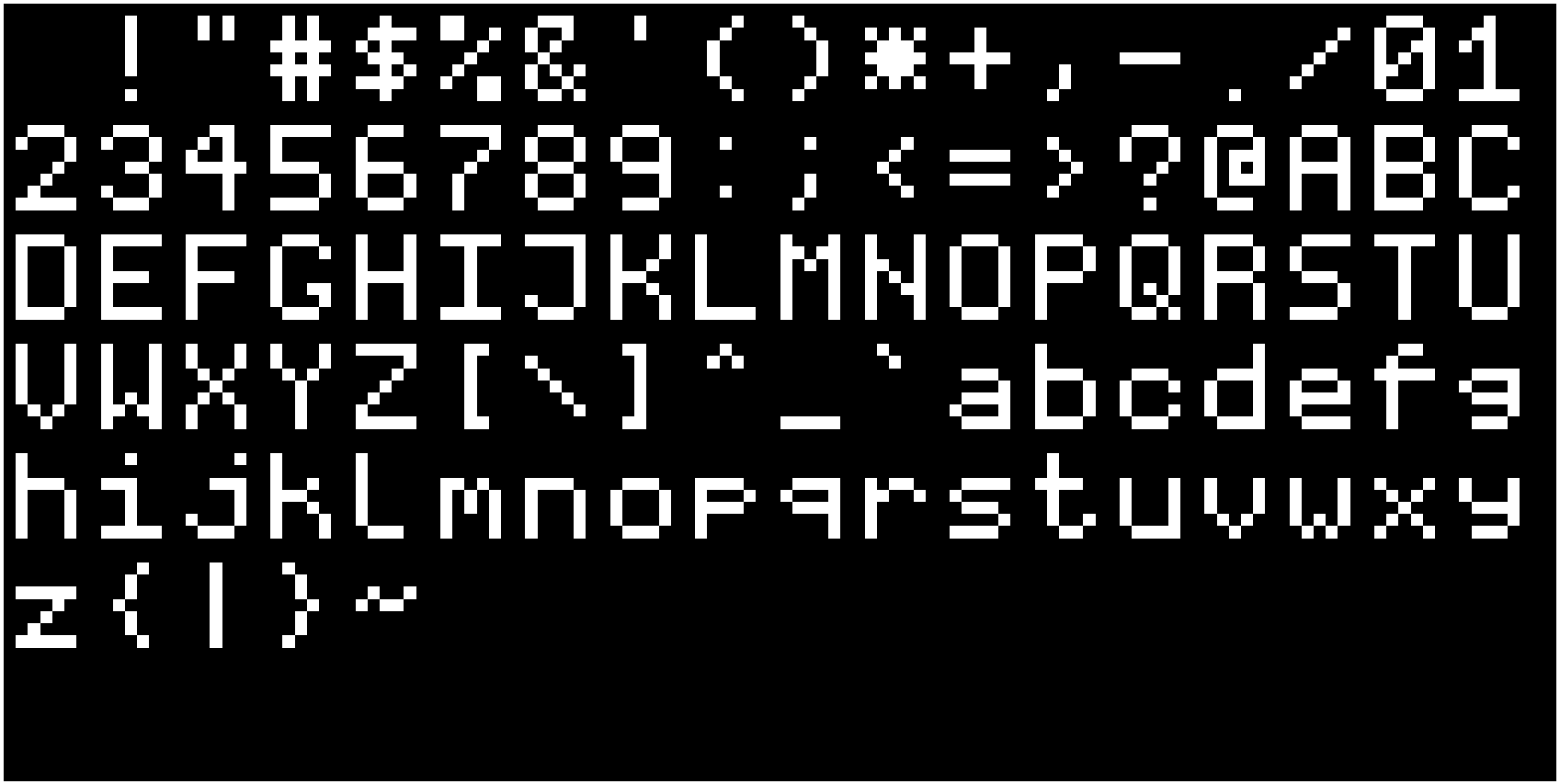
# Рендеринг bitmap-шрифтов

- 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)
- Плохо ведёт себя при масштабировнии (как увеличении, так и уменьшении), mipmap'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)
- Плохо ведёт себя при масштабировнии (как увеличении, так и уменьшении), mipmap'ы не особо помогают
- Очень прост в реализации
- Часто используется для дебажного текста, инди-игр, и т.п.

# Bitmap-шрифт



# Bitmap-шрифт: описание в коде

```
struct bitmap_font
{
  GLuint texture_id;

  struct glyph
  {
    vec2 top_left;
    vec2 bottom_right;
  };

  std::unordered_map<std::uint32_t, glyph_data> glyphs;
};
```

# 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 "О"), 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
  - Запаковка фигур в текстуру, шейдер 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
  - Запаковка фигур в текстуру, шейдер вычисляет площадь пересечения фигуры и пикселя
  - Полигональная аппроксимация глифа (рисуется 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
  - Запаковка фигур в текстуру, шейдер вычисляет площадь пересечения фигуры и пикселя
  - Полигональная аппроксимация глифа (рисуется 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
  - Запаковка фигур в текстуру, шейдер вычисляет площадь пересечения фигуры и пикселя
  - Полигональная аппроксимация глифа (рисуется с использованием stencil буфера) + треугольник со специальным шейдером для каждой кривой Безье
  - Slug algorithm (запатентован)
- Обычно легко переносит масштабирование

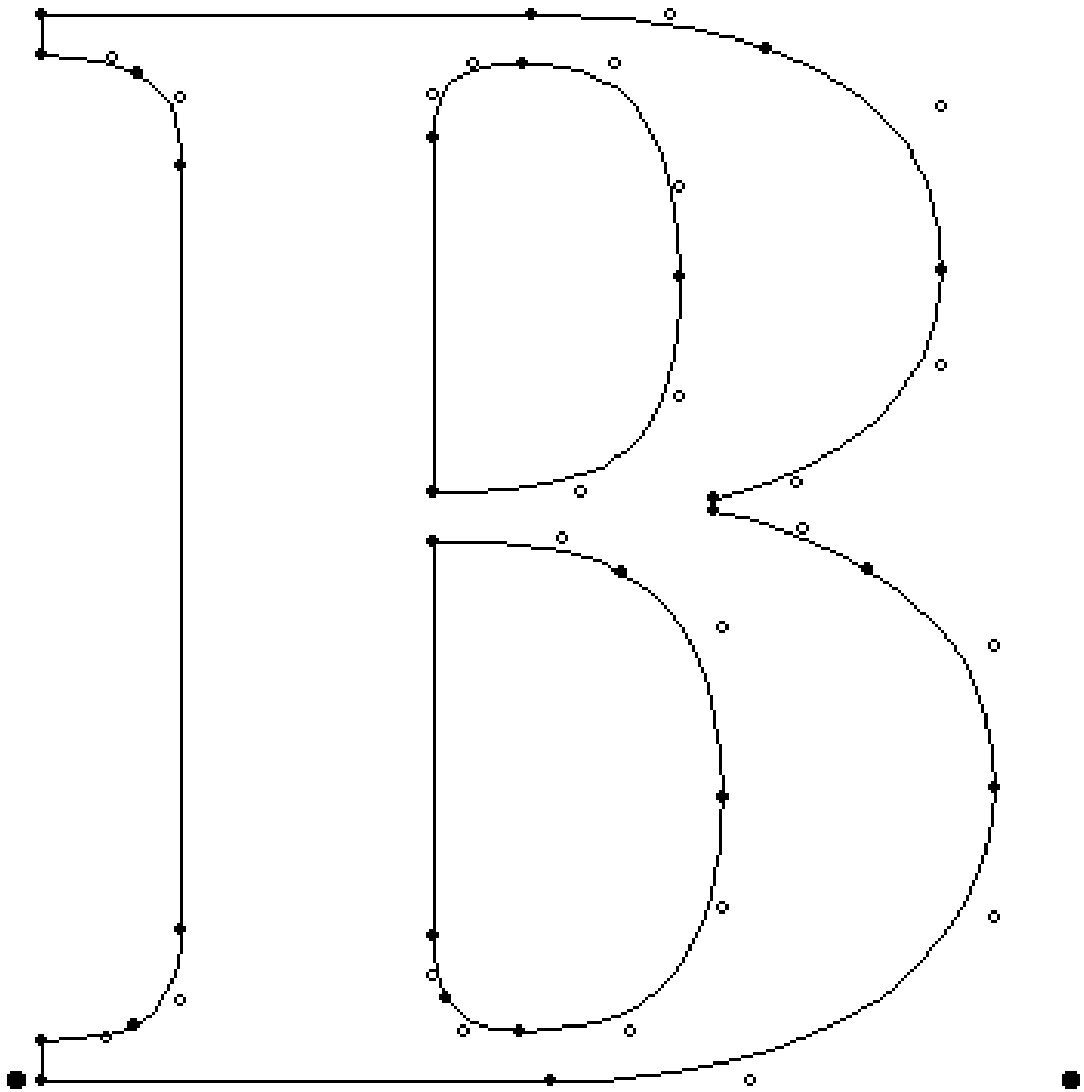
# Рендеринг векторных шрифтов

- 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
  - Запаковка фигур в текстуру, шейдер вычисляет площадь пересечения фигуры и пикселя
  - Полигональная аппроксимация глифа (рисуется 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
  - Запаковка фигур в текстуру, шейдер вычисляет площадь пересечения фигуры и пикселя
  - Полигональная аппроксимация глифа (рисуется 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'а

# 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 (рисуем чёрный пиксель), иначе - нет (рисуем белый пиксель)
  - Обычно добавляется интерполяция от чёрного к белому в районе границы глифа для сглаживания

# 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 (рисуем чёрный пиксель), иначе - нет (рисуем белый пиксель)
  - Обычно добавляется интерполяция от чёрного к белому в районе границы глифа для сглаживания
- 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 (рисуем чёрный пиксель), иначе - нет (рисуем белый пиксель)
  - Обычно добавляется интерполяция от чёрного к белому в районе границы глифа для сглаживания
- Прост в реализации
- Требует чуть больше памяти под текстуры

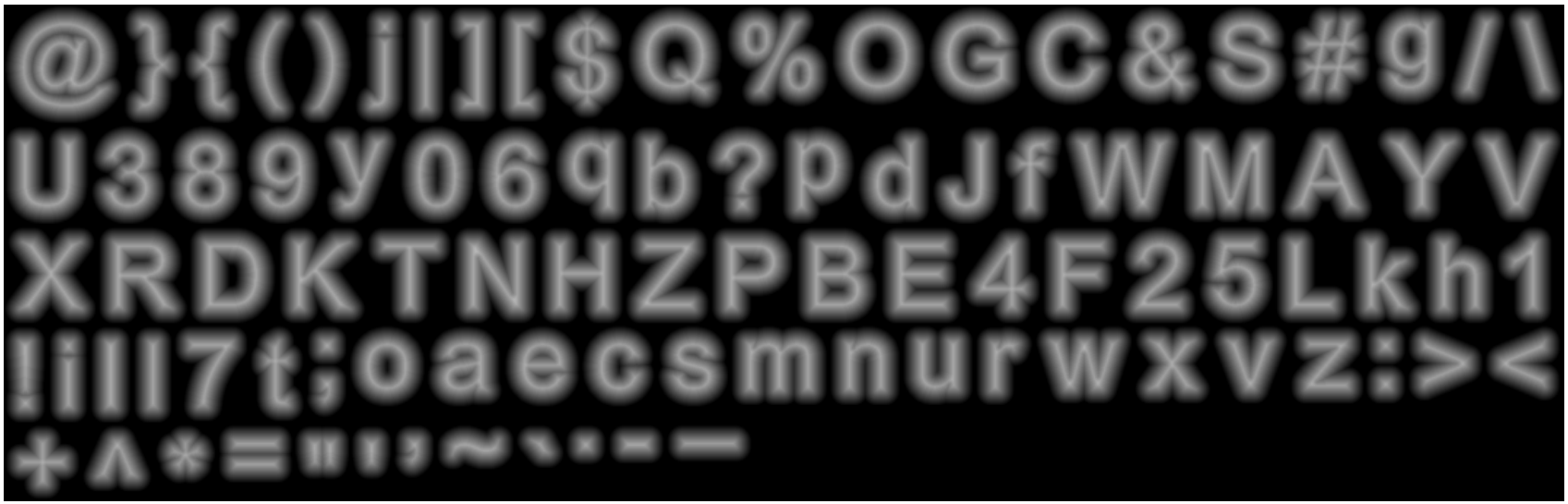
# Рендеринг SDF-шрифтов (Valve, 2007)

- 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 (рисуем чёрный пиксель), иначе - нет (рисуем белый пиксель)
  - Обычно добавляется интерполяция от чёрного к белому в районе границы глифа для сглаживания
- Прост в реализации
- Требует чуть больше памяти под текстуры
- Неплохо масштабируется (бывают артефакты, но куда менее серьёзные, чем для bitmap-шрифтов)

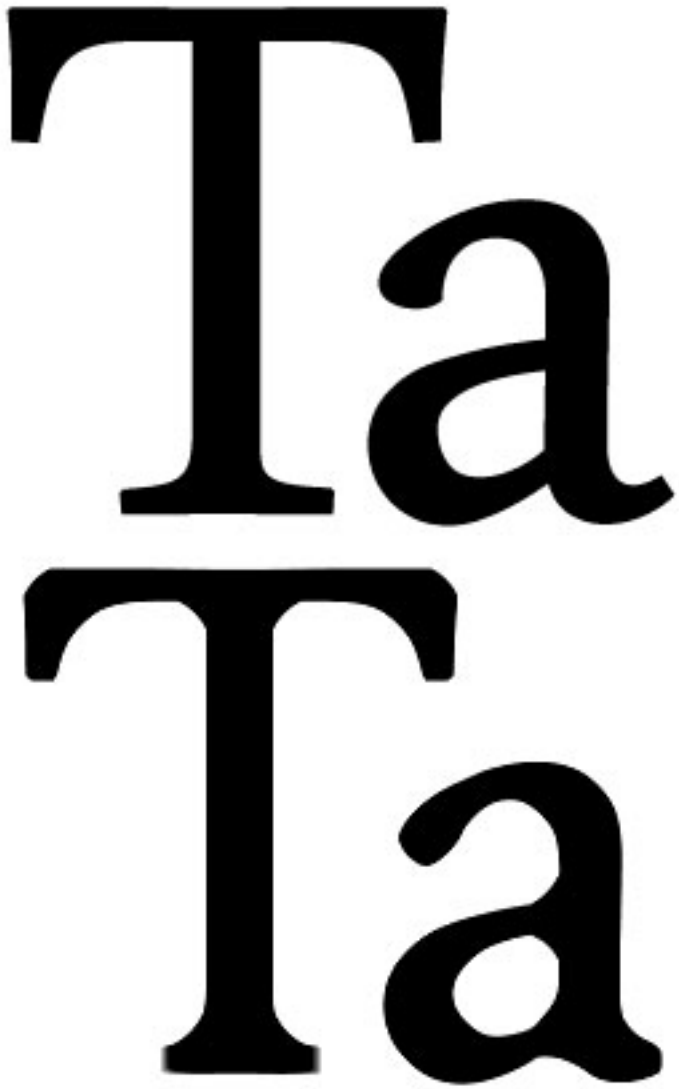
# Рендеринг SDF-шрифтов (Valve, 2007)

- 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 (рисуем чёрный пиксель), иначе - нет (рисуем белый пиксель)
  - Обычно добавляется интерполяция от чёрного к белому в районе границы глифа для сглаживания
- Прост в реализации
- Требует чуть больше памяти под текстуры
- Неплохо масштабируется (бывают артефакты, но куда менее серьёзные, чем для bitmap-шрифтов)
- Один из самых распространённых способов рендеринга шрифтов

# SDF-шрифт


@}{()jl]][$Q%OGC&S#g/\
U389y06qb?pdJfWMAYV
XRDKTNHZPBE4F25Lkh1
!iIl7t;oaecsmnurwxvz:><
+^*=

# SDF-шрифт: артефакты при magnification



# Рендеринг SDF-шрифтов

- Можно легко реализовать много дополнительных эффектов:

# Рендеринг SDF-шрифтов

- Можно легко реализовать много дополнительных эффектов:
  - Обводка текста другим цветом: рисуем цвет обводки, если $0 \leq f(p) \leq \varepsilon$

# Рендеринг SDF-шрифтов

- Можно легко реализовать много дополнительных эффектов:
  - Обводка текста другим цветом: рисуем цвет обводки, если $0 \leq f(p) \leq \varepsilon$
  - Псевдотрёхмерный текст: по градиенту SDF можно восстановить нормаль к глифу

# SDF-шрифт с эффектами

# SDF-шрифт: фрагментный шейдер

```
uniform sampler2D sdfTexture;

in vec2 texcoord;

layout (location = 0) out vec4 out_color;

void main()
{
  float sdfValue = texture(sdfTexture, texcoord).r;
  float alpha = smoothstep(-0.5, 0.5, sdfValue);
  out_color = vec4(0.0, 0.0, 0.0, alpha);
}
```

# Рендеринг текста: ссылки

- FreeType
- harfbuzz
- Туториал по рендерингу bitmap-шрифтов
- Туториал по рендерингу SDF-шрифтов
- Один способ рендеринга векторных шрифтов
- Другой способ рендеринга векторных шрифтов
- Slug algorithm
- Slug library

# Volume rendering

- Рендеринг объектов, заданных распределением свойств в пространстве:

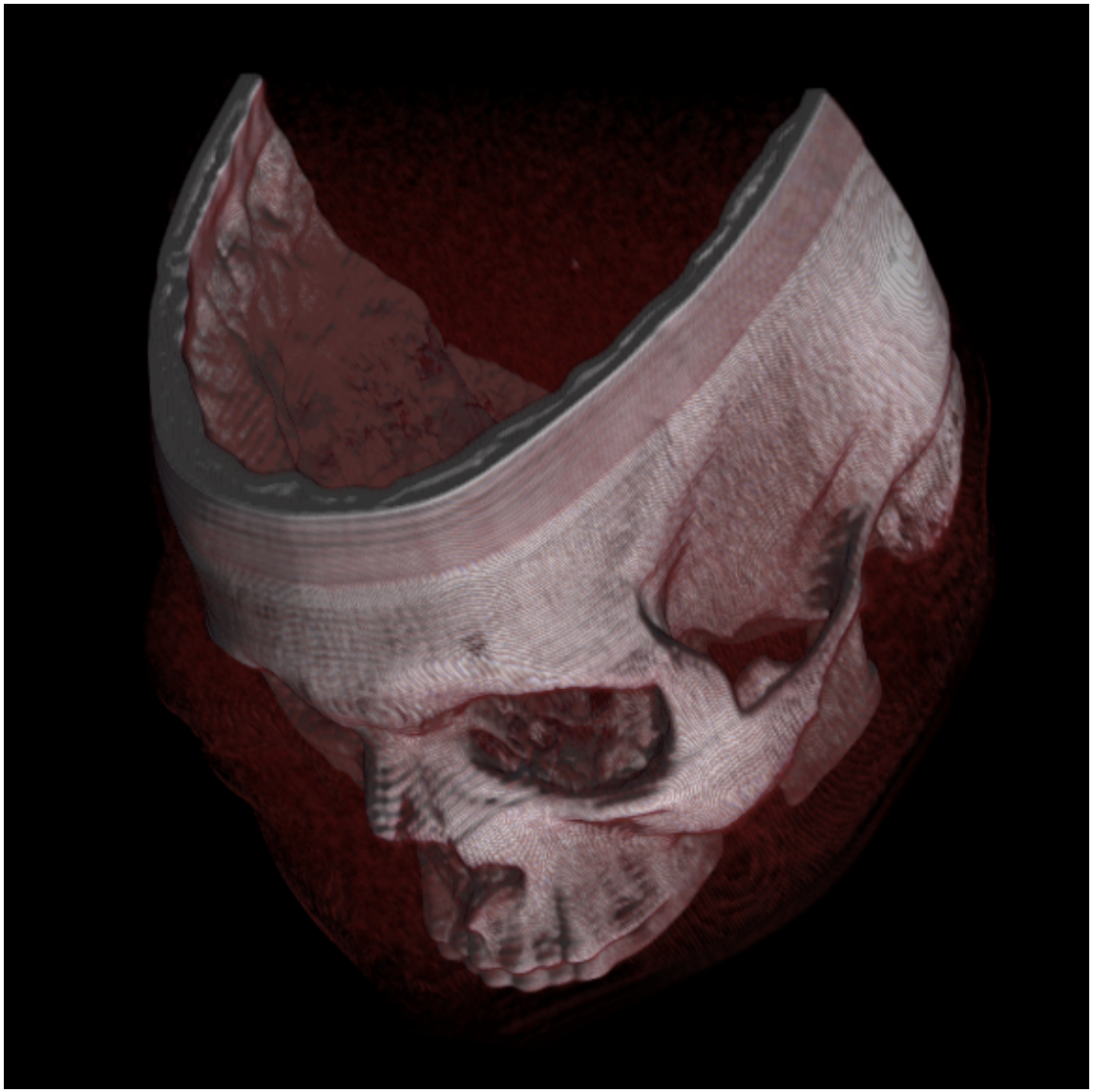
# Volume rendering

- Рендеринг объектов, заданных распределением свойств в пространстве:
  - Цвет
  - Прозрачность
  - Параметры рассеивания

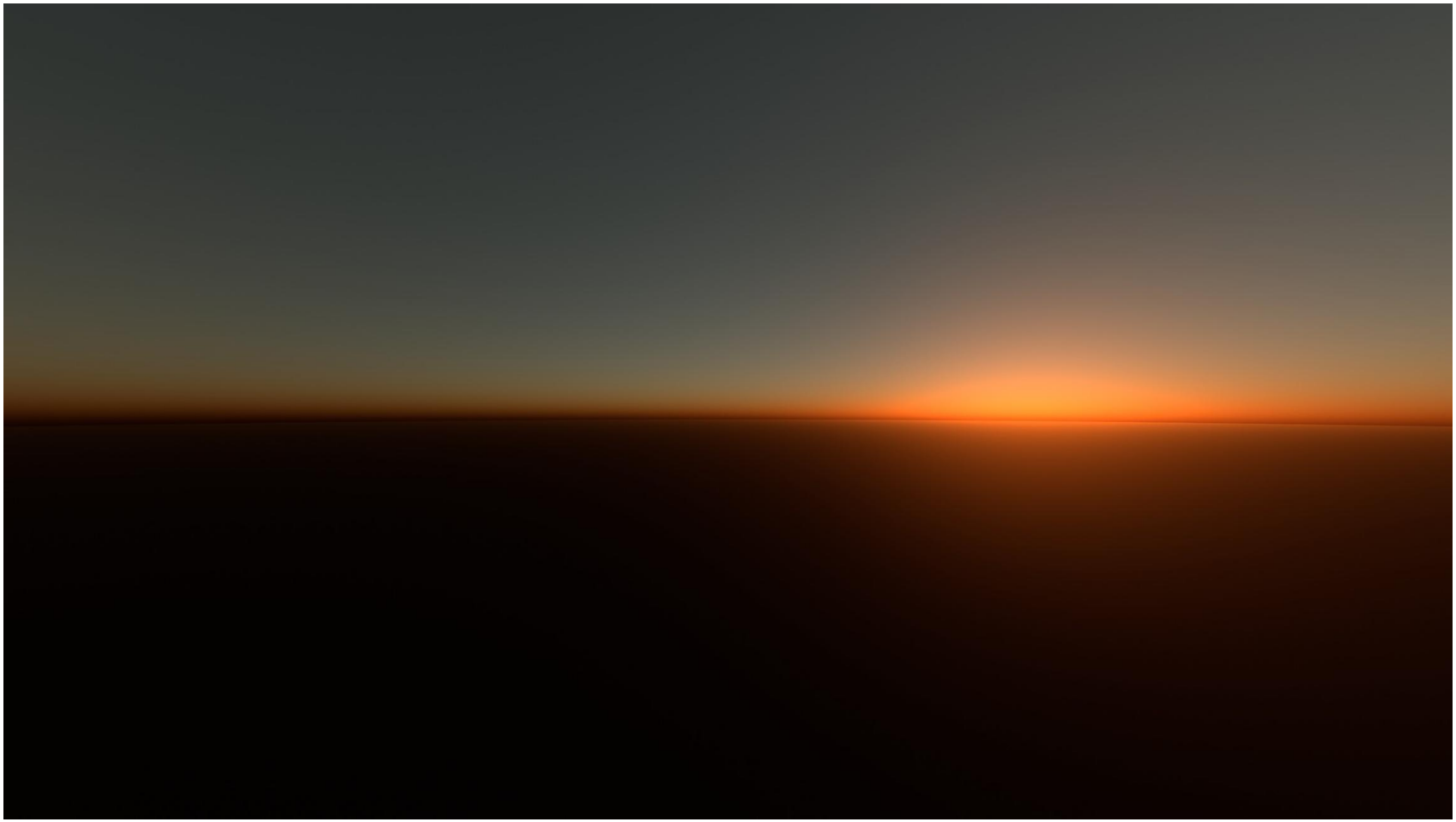
# Volume rendering

- Рендеринг объектов, заданных распределением свойств в пространстве:
  - Цвет
  - Прозрачность
  - Параметры рассеивания
- Медицина: КТ-сканы, МРТ-сканы
- Небо
- Аппроксимации, основанные на теории volume рендеринга: дым, туман, облака, жидкости, subsurface scattering

# Volume-rendering черепа



# Небо (Nishita model)



# Volume rendering: теория

- Свет, попавший в некую трёхмерную среду, может:

# Volume rendering: теория

- Свет, попавший в некую трёхмерную среду, может:
  - Поглотиться (absorption)

# Volume rendering: теория

- Свет, попавший в некую трёхмерную среду, может:
  - Поглотиться (absorption)
  - Рассеяться (scattering), т.е. изменить направление

# Volume rendering: теория

- Свет, попавший в некую трёхмерную среду, может:
  - Поглотиться (absorption)
  - Рассеяться (scattering), т.е. изменить направление
  - Пройти насквозь

# Volume rendering: теория

- Свет, попавший в некую трёхмерную среду, может:
  - Поглотиться (absorption)
  - Рассеяться (scattering), т.е. изменить направление
  - Пройти насквозь
- Кроме того, среда может сама излучать свет (emission)

# Volume rendering: теория

- Свет, попавший в некую трёхмерную среду, может:
  - Поглотиться (absorption)
  - Рассеяться (scattering), т.е. изменить направление
  - Пройти насквозь
- Кроме того, среда может сама излучать свет (emission)
- Absorption + scattering = extinsion

# Volume rendering: теория

- Коэффициенты поглощения $k_a$, рассеяния $k_s$ и излучения $k_e$ могут зависеть от

# Volume rendering: теория

- Коэффициенты поглощения $k_a$, рассеяния $k_s$ и излучения $k_e$ могут зависеть от
  - Точки пространства

# Volume rendering: теория

- Коэффициенты поглощения $k_a$, рассеяния $k_s$ и излучения $k_e$ могут зависеть от
  - Точки пространства
  - Длины волны

# Volume rendering: теория

- Коэффициенты поглощения $k_a$, рассеяния $k_s$ и излучения $k_e$ могут зависеть от
  - Точки пространства
  - Длины волны
  - Угла (рассеяние)

# Volume rendering: теория

- Коэффициенты поглощения $k_a$, рассеяния $k_s$ и излучения $k_e$ могут зависеть от
  - Точки пространства
  - Длины волны
  - Угла (рассеяние)
- $k_a$ и $k_s$ обычно задаются в единицах $м^{-1}$: отношение количества поглощённого/рассеянного света к длине пройденного пути (для бесконечно малых отрезков)

# Volume rendering: теория

- Коэффициенты поглощения $k_a$, рассеяния $k_s$ и излучения $k_e$ могут зависеть от
  - Точки пространства
  - Длины волны
  - Угла (рассеяние)
- $k_a$ и $k_s$ обычно задаются в единицах $м^{-1}$: отношение количества поглощённого/рассеянного света к длине пройденного пути (для бесконечно малых отрезков)
- Аналогично, $k_e$ задаётся в единицах люкс$\cdot м^{-1}$

# Volume rendering: теория

- Коэффициенты поглощения $k_a$, рассеяния $k_s$ и излучения $k_e$ могут зависеть от
  - Точки пространства
  - Длины волны
  - Угла (рассеяние)
- $k_a$ и $k_s$ обычно задаются в единицах $м^{-1}$: отношение количества поглощённого/рассеянного света к длине пройденного пути (для бесконечно малых отрезков)
- Аналогично, $k_e$ задаётся в единицах люкс$\cdot м^{-1}$
- N.B.: часто как $k_e = k_a + k_s$ обозначают коэффициент исчезновения (extinsion)

# Volume rendering: теория

- Пусть есть среда с постоянным коэффициентом поглощения $k_a$, т.е. для бесконечно малого расстояния $\Delta x$ поглотится $k_a \Delta x$ света

# Volume rendering: теория

- Пусть есть среда с постоянным коэффициентом поглощения $k_a$, т.е. для бесконечно малого расстояния $\Delta x$ поглотится $k_a \Delta x$ света
- Как много света поглотится на отрезке длины $L$?

# Volume rendering: теория

- Пусть есть среда с постоянным коэффициентом поглощения $k_a$, т.е. для бесконечно малого расстояния $\Delta x$ поглотится $k_a \Delta x$ света
- Как много света поглотится на отрезке длины $L$?
- Разобьём отрезок на $N$ частей равной длины $\Delta x = \frac{L}{N}$.

# Volume rendering: теория

- Пусть есть среда с постоянным коэффициентом поглощения $k_a$, т.е. для бесконечно малого расстояния $\Delta x$ поглотится $k_a \Delta x$ света
- Как много света поглотится на отрезке длины $L$?
- Разобьём отрезок на $N$ частей равной длины $\Delta x = \frac{L}{N}$.
- На первом отрезке поглотится $k_a \frac{L}{N}$ света, т.е. останется $1 - k_a \frac{L}{N}$. На втором поглотится $k_a \frac{L}{N}$ от того, что осталось после первого отрезка, и всего останется $\left(1 - k_a \frac{L}{N}\right)^2$.

# Volume rendering: теория

- Пусть есть среда с постоянным коэффициентом поглощения $k_a$, т.е. для бесконечно малого расстояния $\Delta x$ поглотится $k_a \Delta x$ света
- Как много света поглотится на отрезке длины $L$?
- Разобьём отрезок на $N$ частей равной длины $\Delta x = \frac{L}{N}$.
- На первом отрезке поглотится $k_a \frac{L}{N}$ света, т.е. останется $1 - k_a \frac{L}{N}$. На втором поглотится $k_a \frac{L}{N}$ от того, что осталось после первого отрезка, и всего останется $\left(1 - k_a \frac{L}{N}\right)^2$.
- Всего после $N$ отрезков останется $\left(1 - k_a \frac{L}{N}\right)^N$ света.

# Volume rendering: теория

- Пусть есть среда с постоянным коэффициентом поглощения $k_a$, т.е. для бесконечно малого расстояния $\Delta x$ поглотится $k_a \Delta x$ света
- Как много света поглотится на отрезке длины $L$?
- Разобьём отрезок на $N$ частей равной длины $\Delta x = \frac{L}{N}$.
- На первом отрезке поглотится $k_a \frac{L}{N}$ света, т.е. останется $1 - k_a \frac{L}{N}$. На втором поглотится $k_a \frac{L}{N}$ от того, что осталось после первого отрезка, и всего останется $\left(1 - k_a \frac{L}{N}\right)^2$.
- Всего после $N$ отрезков останется $\left(1 - k_a \frac{L}{N}\right)^N$ света.
- В пределе: $\lim\limits_{N \to \infty} \left(1 - k_a \frac{L}{N}\right)^N = \exp(-k_a L)$.

# Volume rendering: теория

- Пусть есть среда с постоянным коэффициентом поглощения $k_a$, т.е. для бесконечно малого расстояния $\Delta x$ поглотится $k_a \Delta x$ света
- Как много света поглотится на отрезке длины $L$?
- Разобьём отрезок на $N$ частей равной длины $\Delta x = \frac{L}{N}$.
- На первом отрезке поглотится $k_a \frac{L}{N}$ света, т.е. останется $1 - k_a \frac{L}{N}$. На втором поглотится $k_a \frac{L}{N}$ от того, что осталось после первого отрезка, и всего останется $\left(1 - k_a \frac{L}{N}\right)^2$.
- Всего после $N$ отрезков останется $\left(1 - k_a \frac{L}{N}\right)^N$ света.
- В пределе: $\lim\limits_{N \to \infty} \left(1 - k_a \frac{L}{N}\right)^N = \exp(-k_a L)$.
  - При $L, k_a \in [0, \infty)$ имеем $\exp(-k_a L) \in (0, 1]$

# Volume rendering: теория

- Пусть есть среда с постоянным коэффициентом поглощения $k_a$, т.е. для бесконечно малого расстояния $\Delta x$ поглотится $k_a \Delta x$ света
- Как много света поглотится на отрезке длины $L$?
- Разобьём отрезок на $N$ частей равной длины $\Delta x = \frac{L}{N}$.
- На первом отрезке поглотится $k_a \frac{L}{N}$ света, т.е. останется $1 - k_a \frac{L}{N}$. На втором поглотится $k_a \frac{L}{N}$ от того, что осталось после первого отрезка, и всего останется $\left(1 - k_a \frac{L}{N}\right)^2$.
- Всего после $N$ отрезков останется $\left(1 - k_a \frac{L}{N}\right)^N$ света.
- В пределе: $\lim\limits_{N \to \infty} \left(1 - k_a \frac{L}{N}\right)^N = \exp(-k_a L)$.
  - При $L, k_a \in [0, \infty)$ имеем $\exp(-k_a L) \in (0, 1]$
  - Если $L = 0$, то весь свет останется (ничего не поглотится)
  - Если $L \to \infty$, то весь свет поглотится

# Volume rendering: теория

- Пусть есть среда с постоянным коэффициентом поглощения $k_a$ и излучения $k_e$, т.е. для бесконечно малого расстояния $\Delta x$ излучится $k_e \Delta x$ света

# Volume rendering: теория

- Пусть есть среда с постоянным коэффициентом поглощения $k_a$ и излучения $k_e$, т.е. для бесконечно малого расстояния $\Delta x$ излучится $k_e \Delta x$ света
- Как много света излучится на отрезке длины $L$?

# Volume rendering: теория

- Пусть есть среда с постоянным коэффициентом поглощения $k_a$ и излучения $k_e$, т.е. для бесконечно малого расстояния $\Delta x$ излучится $k_e \Delta x$ света
- Как много света излучится на отрезке длины $L$?
- Разобьём отрезок на $N$ частей равной длины $\Delta x = \frac{L}{N}$.

# Volume rendering: теория

- Пусть есть среда с постоянным коэффициентом поглощения $k_a$ и излучения $k_e$, т.е. для бесконечно малого расстояния $\Delta x$ излучится $k_e \Delta x$ света
- Как много света излучится на отрезке длины $L$?
- Разобьём отрезок на $N$ частей равной длины $\Delta x = \frac{L}{N}$.
- На первом отрезке излучится $k_e \Delta x$ света. На втором отрезке излучится $k_e \Delta x$ света, плюс $(1 - k_a \Delta x) k_e \Delta x$ от первого отрезка, в сумме $(1 + (1 - k_a \Delta x))\, k_e \Delta x$

# Volume rendering: теория

- Пусть есть среда с постоянным коэффициентом поглощения $k_a$ и излучения $k_e$, т.е. для бесконечно малого расстояния $\Delta x$ излучится $k_e \Delta x$ света
- Как много света излучится на отрезке длины $L$?
- Разобьём отрезок на $N$ частей равной длины $\Delta x = \frac{L}{N}$.
- На первом отрезке излучится $k_e \Delta x$ света. На втором отрезке излучится $k_e \Delta x$ света, плюс $(1 - k_a \Delta x) k_e \Delta x$ от первого отрезка, в сумме $(1 + (1 - k_a \Delta x))\, k_e \Delta x$
- На третьем отрезке в сумме $\left(1 + (1 - k_a \Delta x) + (1 - k_a \Delta x)^2\right) k_e \Delta x$

# Volume rendering: теория

- Пусть есть среда с постоянным коэффициентом поглощения $k_a$ и излучения $k_e$, т.е. для бесконечно малого расстояния $\Delta x$ излучится $k_e \Delta x$ света
- Как много света излучится на отрезке длины $L$?
- Разобьём отрезок на $N$ частей равной длины $\Delta x = \frac{L}{N}$.
- На первом отрезке излучится $k_e \Delta x$ света. На втором отрезке излучится $k_e \Delta x$ света, плюс $(1 - k_a \Delta x) k_e \Delta x$ от первого отрезка, в сумме $(1 + (1 - k_a \Delta x))\, k_e \Delta x$
- На третьем отрезке в сумме $\left(1 + (1 - k_a \Delta x) + (1 - k_a \Delta x)^2\right) k_e \Delta x$
- На $N$ отрезках: $\frac{1-(1-k_a \Delta x)^N}{k_a \Delta x} k_e \Delta x = \frac{1-(1-k_a \Delta x)^N}{k_a} k_e$

# Volume rendering: теория

- Пусть есть среда с постоянным коэффициентом поглощения $k_a$ и излучения $k_e$, т.е. для бесконечно малого расстояния $\Delta x$ излучится $k_e \Delta x$ света
- Как много света излучится на отрезке длины $L$?
- Разобьём отрезок на $N$ частей равной длины $\Delta x = \frac{L}{N}$.
- На первом отрезке излучится $k_e \Delta x$ света. На втором отрезке излучится $k_e \Delta x$ света, плюс $(1 - k_a \Delta x) k_e \Delta x$ от первого отрезка, в сумме $(1 + (1 - k_a \Delta x))\, k_e \Delta x$
- На третьем отрезке в сумме $\left(1 + (1 - k_a \Delta x) + (1 - k_a \Delta x)^2\right) k_e \Delta x$
- На $N$ отрезках: $\frac{1-(1-k_a \Delta x)^N}{k_a \Delta x} k_e \Delta x = \frac{1-(1-k_a \Delta x)^N}{k_a} k_e$
- В пределе: $\lim\limits_{N \to \infty} \frac{1-(1-k_a \Delta x)^N}{k_a} k_e = \frac{1-\exp(-k_a L)}{k_a} k_e$.

# Volume rendering: теория

- Пусть есть среда с постоянным коэффициентом поглощения $k_a$ и излучения $k_e$, т.е. для бесконечно малого расстояния $\Delta x$ излучится $k_e \Delta x$ света
- Как много света излучится на отрезке длины $L$?
- Разобьём отрезок на $N$ частей равной длины $\Delta x = \frac{L}{N}$.
- На первом отрезке излучится $k_e \Delta x$ света. На втором отрезке излучится $k_e \Delta x$ света, плюс $(1 - k_a \Delta x) k_e \Delta x$ от первого отрезка, в сумме $(1 + (1 - k_a \Delta x))\, k_e \Delta x$
- На третьем отрезке в сумме $\left(1 + (1 - k_a \Delta x) + (1 - k_a \Delta x)^2\right) k_e \Delta x$
- На $N$ отрезках: $\frac{1-(1-k_a \Delta x)^N}{k_a \Delta x} k_e \Delta x = \frac{1-(1-k_a \Delta x)^N}{k_a} k_e$
- В пределе: $\lim\limits_{N \to \infty} \frac{1-(1-k_a \Delta x)^N}{k_a} k_e = \frac{1-\exp(-k_a L)}{k_a} k_e$.
  - Сингулярность при $k_a = 0$, в пределе получим $L k_e$

# Volume rendering: теория

- Пусть есть среда с коэффициентом поглощения, зависящим от точки $k_a(x)$

# Volume rendering: теория

- Пусть есть среда с коэффициентом поглощения, зависящим от точки $k_a(x)$
- Как много света поглотится на отрезке длины $L$?

# Volume rendering: теория

- Пусть есть среда с коэффициентом поглощения, зависящим от точки $k_a(x)$
- Как много света поглотится на отрезке длины $L$?
- Разобьём отрезок на $N$ частей равной длины $\Delta x = \frac{L}{N}$, и пусть $x_i$ - центр i-ого отрезка

# Volume rendering: теория

- Пусть есть среда с коэффициентом поглощения, зависящим от точки $k_a(x)$
- Как много света поглотится на отрезке длины $L$?
- Разобьём отрезок на $N$ частей равной длины $\Delta x = \frac{L}{N}$, и пусть $x_i$ - центр i-ого отрезка
- i-ый отрезок поглотит примерно $1 - \exp\left(-k_a(x_i)\Delta x\right)$ света и оставит $\exp\left(-k_a(x_i)\Delta x\right)$

# Volume rendering: теория

- Пусть есть среда с коэффициентом поглощения, зависящим от точки $k_a(x)$
- Как много света поглотится на отрезке длины $L$?
- Разобьём отрезок на $N$ частей равной длины $\Delta x = \frac{L}{N}$, и пусть $x_i$ - центр i-ого отрезка
- i-ый отрезок поглотит примерно $1 - \exp\left(-k_a(x_i)\Delta x\right)$ света и оставит $\exp\left(-k_a(x_i)\Delta x\right)$
- В сумме останется $\prod \exp\left(-k_a(x_i)\Delta x\right) = \exp\left(-\sum k_a(x_i)\Delta x\right)$

# Volume rendering: теория

- Пусть есть среда с коэффициентом поглощения, зависящим от точки $k_a(x)$
- Как много света поглотится на отрезке длины $L$?
- Разобьём отрезок на $N$ частей равной длины $\Delta x = \frac{L}{N}$, и пусть $x_i$ - центр i-ого отрезка
- i-ый отрезок поглотит примерно $1 - \exp\left(-k_a(x_i)\Delta x\right)$ света и оставит $\exp\left(-k_a(x_i)\Delta x\right)$
- В сумме останется $\prod \exp\left(-k_a(x_i)\Delta x\right) = \exp\left(-\sum k_a(x_i)\Delta x\right)$
- В пределе: $\exp\left(-\int\limits_0^L k_a(x)dx\right)$

# Volume rendering: теория

- Пусть есть среда с коэффициентом поглощения, зависящим от точки $k_a(x)$
- Как много света поглотится на отрезке длины $L$?
- Разобьём отрезок на $N$ частей равной длины $\Delta x = \frac{L}{N}$, и пусть $x_i$ - центр i-ого отрезка
- i-ый отрезок поглотит примерно $1 - \exp\left(-k_a(x_i)\Delta x\right)$ света и оставит $\exp\left(-k_a(x_i)\Delta x\right)$
- В сумме останется $\prod \exp\left(-k_a(x_i)\Delta x\right) = \exp\left(-\sum k_a(x_i)\Delta x\right)$
- В пределе: $\exp\left(-\int\limits_0^L k_a(x)dx\right)$
- Величина $\int\limits_0^L k_a(x)dx$ называется *оптической глубиной (optical depth)*

# Volume rendering: теория

- Пусть есть среда с коэффициентами поглощения и излучения, зависящими от точки $k_a(x), k_e(x)$

# Volume rendering: теория

- Пусть есть среда с коэффициентами поглощения и излучения, зависящими от точки $k_a(x), k_e(x)$
- Как много света излучится на отрезке длины $L$?

# Volume rendering: теория

- Пусть есть среда с коэффициентами поглощения и излучения, зависящими от точки $k_a(x), k_e(x)$
- Как много света излучится на отрезке длины $L$?
- $\int_0^L \exp\left(-\int_x^L k_a(s)ds\right) k_e(x)dx$

# Volume rendering: рассеяние (scattering)

- Обычно любая среда рассеивает свет в определённом направлении

# Volume rendering: рассеяние (scattering)

- Обычно любая среда рассеивает свет в определённом направлении
- Phase function: описывает количество света, рассеянного под определённым углом (относительно падающего света) $f(x, \lambda, \theta)$

# Volume rendering: рассеяние (scattering)

- Обычно любая среда рассеивает свет в определённом направлении
- Phase function: описывает количество света, рассеянного под определённым углом (относительно падающего света) $f(x, \lambda, \theta)$
- Зависит от конкретных составляющих среды, выводится из решения уравнений Максвелла

# Рассеяние Релея (Rayleigh)

- Рассеяние света частицами намного меньшими длины волны (например, молекулами газов $N_2$ и $O_2$)

# Рассеяние Релея (Rayleigh)

- Рассеяние света частицами намного меньшими длины волны (например, молекулами газов $N_2$ и $O_2$)
- $f \cong \frac{1+\cos(\theta)^2}{\lambda^4}$

# Рассеяние Релея (Rayleigh)

- Рассеяние света частицами намного меньшими длины волны (например, молекулами газов $N_2$ и $O_2$)
- $f \cong \frac{1+\cos(\theta)^2}{\lambda^4}$
- Больше рассеяния вперёд и назад, чем в стороны

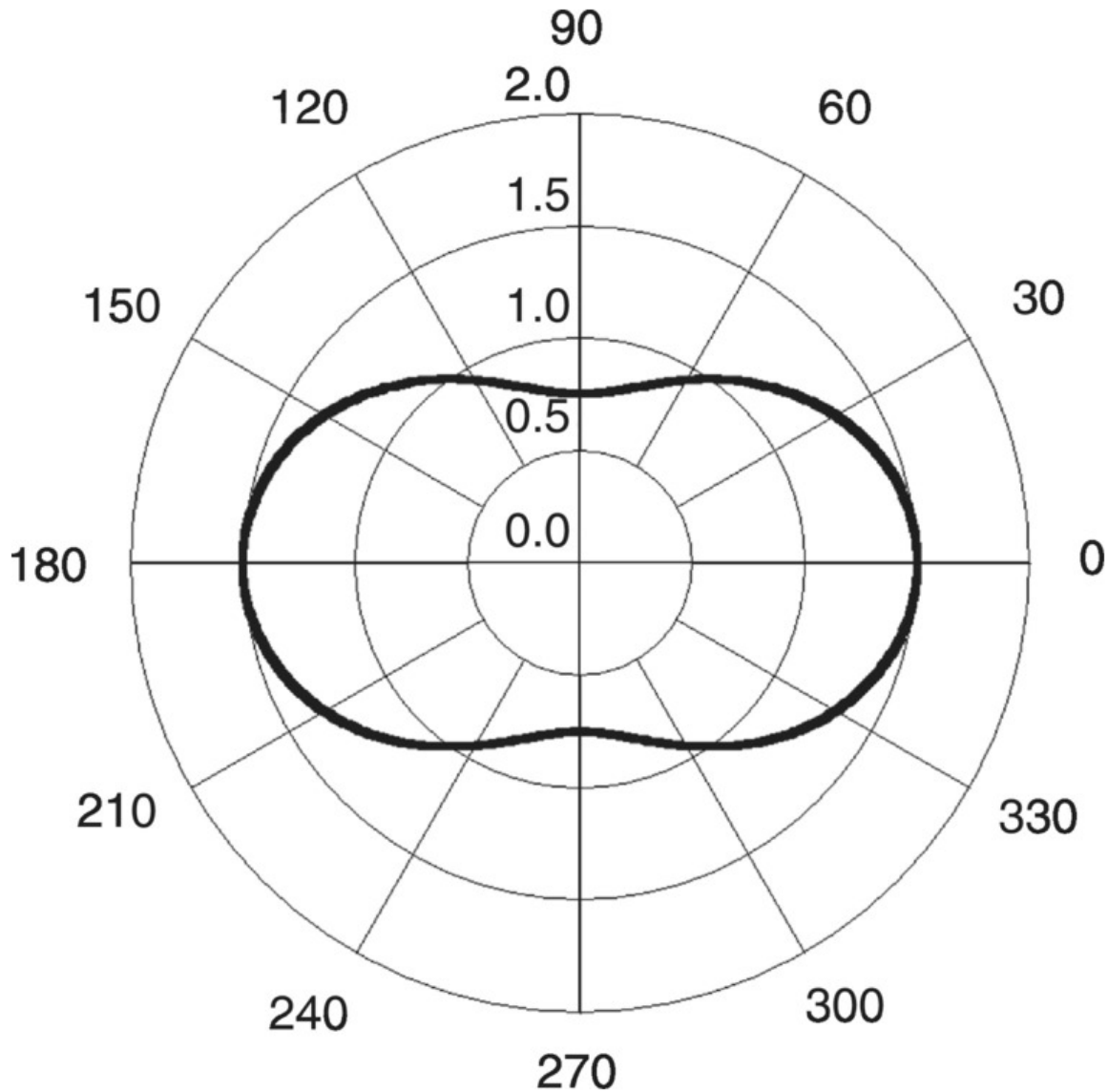
# Рассеяние Релея (Rayleigh)

- Рассеяние света частицами намного меньшими длины волны (например, молекулами газов $N_2$ и $O_2$)
- $f \cong \frac{1+\cos(\theta)^2}{\lambda^4}$
- Больше рассеяния вперёд и назад, чем в стороны
- Короткие волны (e.g. синие) рассеиваются намного больше длинных (e.g. красных)

# Рассеяние Релея (Rayleigh)

- Рассеяние света частицами намного меньшими длины волны (например, молекулами газов $N_2$ и $O_2$)
- $f \cong \frac{1+\cos(\theta)^2}{\lambda^4}$
- Больше рассеяния вперёд и назад, чем в стороны
- Короткие волны (e.g. синие) рассеиваются намного больше длинных (e.g. красных)
- Главный эффект, ответственный за цвет неба

# Рассеяние Релея: phase function



# Рассеяние Ми (Mie)

- Рассеяние света частицами сравнимыми по размеру с длиной волны (например, каплями воды)

# Рассеяние Ми (Mie)

- Рассеяние света частицами сравнимыми по размеру с длиной волны (например, каплями воды)
- Точная формула - разложение в ряд по сферическим гармоникам

# Рассеяние Ми (Mie)

- Рассеяние света частицами сравнимыми по размеру с длиной волны (например, каплями воды)
- Точная формула - разложение в ряд по сферическим гармоникам
- Обычно для рендеринга используются аппроксимации

# Рассеяние Ми (Mie)

- Рассеяние света частицами сравнимыми по размеру с длиной волны (например, каплями воды)
- Точная формула - разложение в ряд по сферическим гармоникам
- Обычно для рендеринга используются аппроксимации
- Влияет на цвет неба (частицы воды, аэрозоли) и облаков

# Volume rendering: рассеяние

- Значительно усложняет уравнения: свет, пришедший из любого направления, может рассеяться в сторону камеры в любой точке

# Volume rendering: рассеяние

- Значительно усложняет уравнения: свет, пришедший из любого направления, может рассеяться в сторону камеры в любой точке
- Различают
  - Zero-scattering: рассеяние не учитывается
  - Single scattering: учитывается свет, пришедший из источника света и рассеявшийся сразу в направлении камеры (самое часто используемое приближение)
  - Multiple scattering: свет, пришедший из источника света, может рассеяться много раз и в итоге прийти в камеру

# Volume rendering: применения

- Для визуализации трёхмерных данных (МРТ-сканы и т.п.) обычно игнорируют рассеяние, и считают среду полупрозрачной и излучающей свет

# Volume rendering: применения

- Для визуализации трёхмерных данных (МРТ-сканы и т.п.) обычно игнорируют рассеяние, и считают среду полупрозрачной и излучающей свет
  - Вместо $k_a$ часто хранят некую "непрозрачность"(как значение альфа-канала), не имеющую физического смысла

# Volume rendering: применения

- Для визуализации трёхмерных данных (МРТ-сканы и т.п.) обычно игнорируют рассеяние, и считают среду полупрозрачной и излучающей свет
  - Вместо $k_a$ часто хранят некую "непрозрачность"(как значение альфа-канала), не имеющую физического смысла
  - Цвет излучаемого света может храниться в текстуре, а может зависеть от значения альфа-канала (transfer function)

# Volume rendering: применения

- Для визуализации трёхмерных данных (МРТ-сканы и т.п.) обычно игнорируют рассеяние, и считают среду полупрозрачной и излучающей свет
  - Вместо $k_a$ часто хранят некую "непрозрачность"(как значение альфа-канала), не имеющую физического смысла
  - Цвет излучаемого света может храниться в текстуре, а может зависеть от значения альфа-канала (transfer function)
- Для визуализации неба, облаков и т.д. рассеяние - обязательная составляющая, коэффициенты часто берут из реальных данных

# Volume rendering: представление данных

- 3D текстура

# Volume rendering: представление данных

- 3D текстура
- Явная функция

# Volume rendering: представление данных

- 3D текстура
- Явная функция
- Их комбинация

# Volume rendering: методы

- Slicing
- Splatting
- Raymarching

# Volume rendering: slicing

- Объект "разрезается"набором равноудалённых плоскостей, перпендикулярных направлению на камеру (чем больше, тем дороже и точнее)

# Volume rendering: slicing

- Объект "разрезается"набором равноудалённых плоскостей, перпендикулярных направлению на камеру (чем больше, тем дороже и точнее)
- Каждый слой (slice) рисуется как текстурированный многоугольник, читая данные из 3D текстуры или из явной фукнции, задающей объект

# Volume rendering: slicing

- Объект "разрезается"набором равноудалённых плоскостей, перпендикулярных направлению на камеру (чем больше, тем дороже и точнее)
- Каждый слой (slice) рисуется как текстурированный многоугольник, читая данные из 3D текстуры или из явной фукнции, задающей объект
- Слои накладываются друг на друга с blending'ом

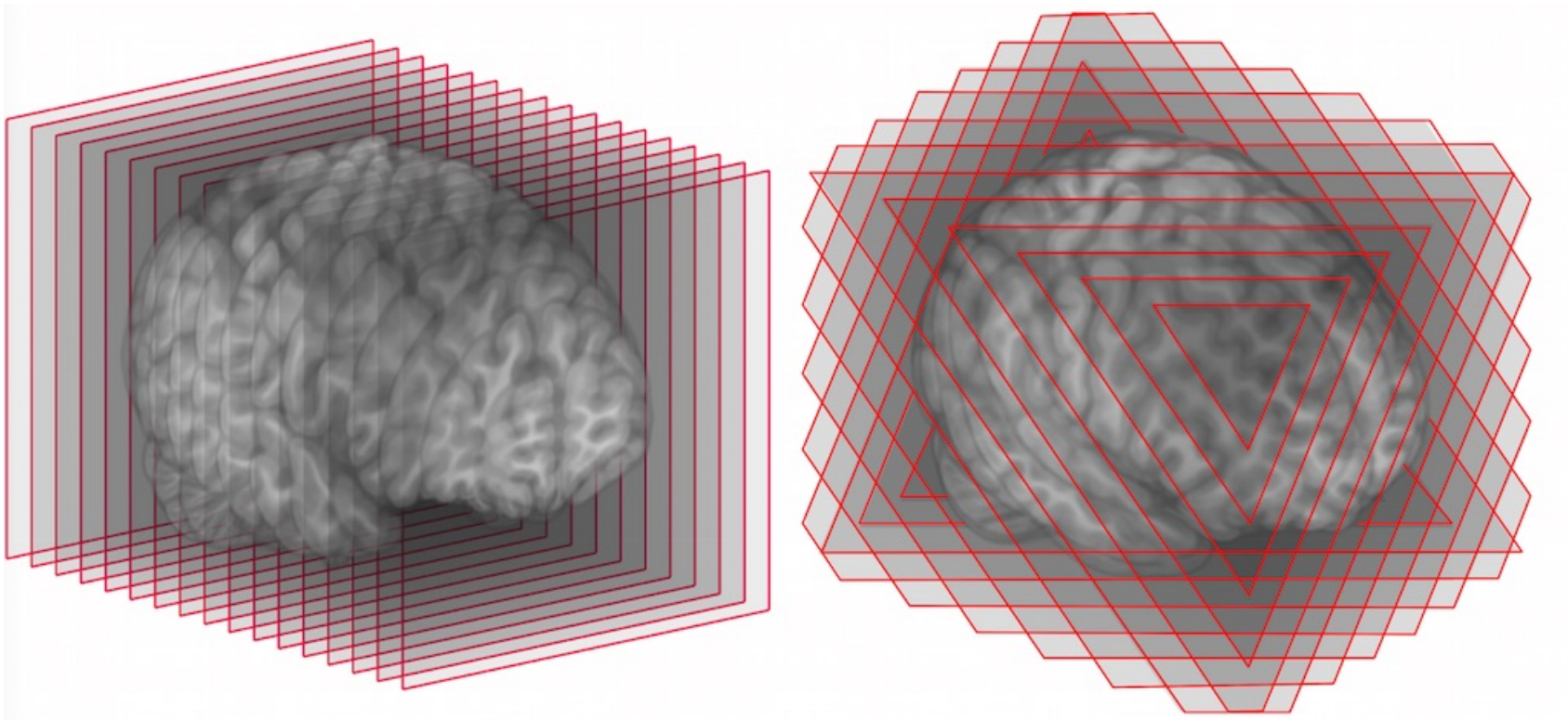
# Volume rendering: slicing

- Объект "разрезается"набором равноудалённых плоскостей, перпендикулярных направлению на камеру (чем больше, тем дороже и точнее)
- Каждый слой (slice) рисуется как текстурированный многоугольник, читая данные из 3D текстуры или из явной фукнции, задающей объект
- Слои накладываются друг на друга с blending'ом
- Нужно отсортировать слои от дальнего к ближнему (back-to-front) или наоборот (front-to-back), об этого будет зависеть режим blending'а

# Volume rendering: slicing

- Объект "разрезается"набором равноудалённых плоскостей, перпендикулярных направлению на камеру (чем больше, тем дороже и точнее)
- Каждый слой (slice) рисуется как текстурированный многоугольник, читая данные из 3D текстуры или из явной фукнции, задающей объект
- Слои накладываются друг на друга с blending'ом
- Нужно отсортировать слои от дальнего к ближнему (back-to-front) или наоборот (front-to-back), об этого будет зависеть режим blending'а
- В значении альфа-канала нужно учесть расстояние между слоями

# Volume rendering: slicing



# Volume rendering: slicing

- Нужно обновлять геометрию slice'ов при каждом изменении камеры

# Volume rendering: slicing

- Нужно обновлять геометрию slice'ов при каждом изменении камеры(это проще, чем звучит)

# Volume rendering: slicing

- Нужно обновлять геометрию slice'ов при каждом изменении камеры(это проще, чем звучит)
- Использует встроенный механизм (blending) $\Rightarrow$ работает достаточно быстро

# Volume rendering: slicing

- Нужно обновлять геометрию slice'ов при каждом изменении камеры(это проще, чем звучит)
- Использует встроенный механизм (blending) $\Rightarrow$ работает достаточно быстро
- Тяжело учесть освещение/рассеяние

# Volume rendering: slicing

- Нужно обновлять геометрию slice'ов при каждом изменении камеры(это проще, чем звучит)
- Использует встроенный механизм (blending) $\Rightarrow$ работает достаточно быстро
- Тяжело учесть освещение/рассеяние
- Часто применяется в медицине

# Volume rendering: splatting

- Объект рисуется как набор частиц (splats) внутри объёма объекта

# Volume rendering: splatting

- Объект рисуется как набор частиц (splats) внутри объёма объекта
- Частицы рисуются как прямоугольники, перпендикулярные направлению на камеру (billboards), с gaussian-текстурой и значениями цвета/прозрачности, взятыми из центра частицы

# Volume rendering: splatting

- Объект рисуется как набор частиц (splats) внутри объёма объекта
- Частицы рисуются как прямоугольники, перпендикулярные направлению на камеру (billboards), с gaussian-текстурой и значениями цвета/прозрачности, взятыми из центра частицы
- Так же, как в slicing'е, нужно отсортировать частицы, правильно настроить blending и значение альфа-канала

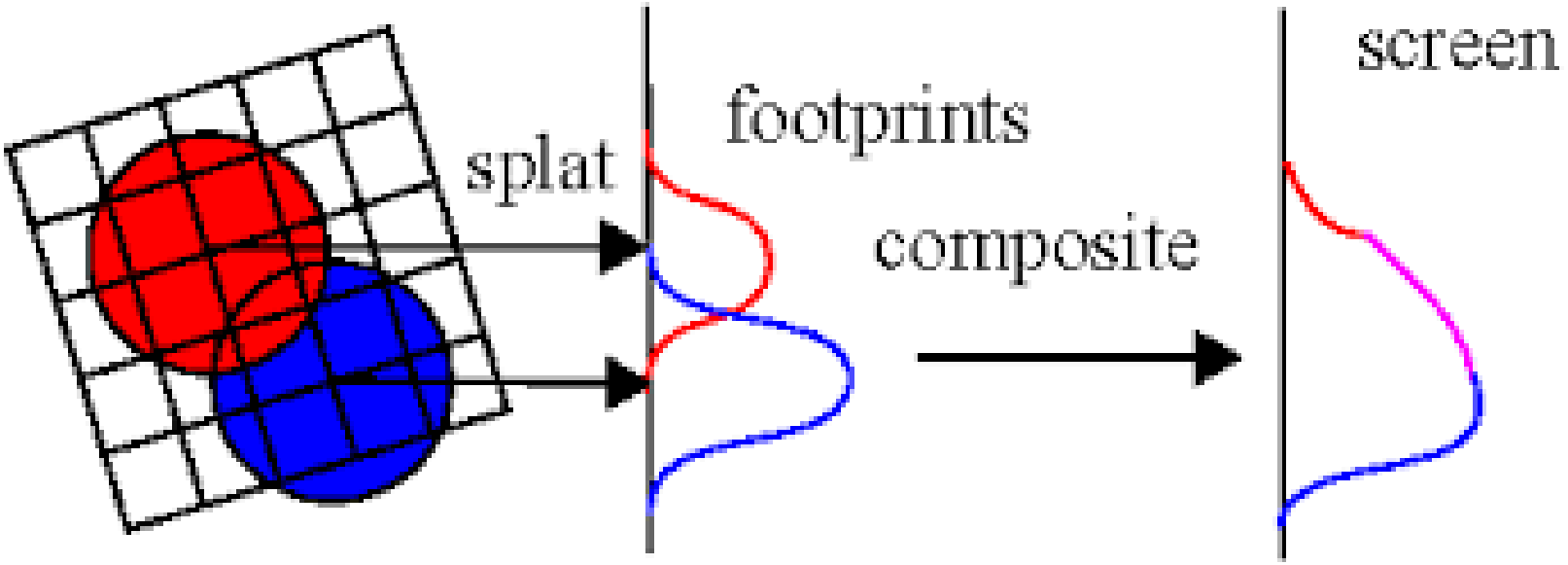
# Volume rendering: splatting

- Объект рисуется как набор частиц (splats) внутри объёма объекта
- Частицы рисуются как прямоугольники, перпендикулярные направлению на камеру (billboards), с gaussian-текстурой и значениями цвета/прозрачности, взятыми из центра частицы
- Так же, как в slicing'е, нужно отсортировать частицы, правильно настроить blending и значение альфа-канала
- Не нужно менять геометрию каждый кадр, но нужно сортировать частицы

# Volume rendering: splatting

- Объект рисуется как набор частиц (splats) внутри объёма объекта
- Частицы рисуются как прямоугольники, перпендикулярные направлению на камеру (billboards), с gaussian-текстурой и значениями цвета/прозрачности, взятыми из центра частицы
- Так же, как в slicing'е, нужно отсортировать частицы, правильно настроить blending и значение альфа-канала
- Не нужно менять геометрию каждый кадр, но нужно сортировать частицы
- Сложнее добиться красивой картинки при малом числе частиц

# Volume rendering: splatting



# Ray*

Есть несколько алгоритмов, название которых звучит как ray-<глагол>:

# Ray*

Есть несколько алгоритмов, название которых звучит как ray-<глагол>:

- Raycasting - алгоритм, использовавшийся в старых 3D играх: находится пересечение луча из камеры в конкретном направлении в плоскости XZ со стенами сцены; по расстоянию до пересечения можно вычислить, как рисовать целый столбец пикселей с одинаковой экранной X-координатой

# Ray*

Есть несколько алгоритмов, название которых звучит как ray-<глагол>:

- Raycasting - алгоритм, использовавшийся в старых 3D играх: находится пересечение луча из камеры в конкретном направлении в плоскости XZ со стенами сцены; по расстоянию до пересечения можно вычислить, как рисовать целый столбец пикселей с одинаковой экранной X-координатой
- Raytracing - алгоритм, использующийся для фотореалистичного рендеринга: находится пересечение луча из камеры с ближайшим объектом сцены (обычно используя вспомогательные структуры данных для ускорения), после чего из точки пересечения рекурсивно бросаются новые лучи

# Ray*

Есть несколько алгоритмов, название которых звучит как ray-<глагол>:

- Raycasting - алгоритм, использовавшийся в старых 3D играх: находится пересечение луча из камеры в конкретном направлении в плоскости XZ со стенами сцены; по расстоянию до пересечения можно вычислить, как рисовать целый столбец пикселей с одинаковой экранной X-координатой
- Raytracing - алгоритм, использующийся для фотореалистичного рендеринга: находится пересечение луча из камеры с ближайшим объектом сцены (обычно используя вспомогательные структуры данных для ускорения), после чего из точки пересечения рекурсивно бросаются новые лучи
- Raymarching - алгоритм, использующийся для volume рендеринга, screen-space отражений, SDF, и подобного: вдоль луча из камеры выбирается дискретный набор точек (часто - с фиксированным шагом), в этих точках вычисляются значения некоторой функции и (в случае volume рендеринга) по этим точкам вычисляется искомый интеграл

# Raycasting

# Volume rendering: raymarching

- Рисуется некая грубая оболочка объекта, часто - AABB (axis-aligned bounding box), на неё удобно накладывать 3D-текстуру
  - Можно с **front-face culling**, чтобы алгоритм работал даже в случае, если камера находится внутри объекта

# Volume rendering: raymarching

- Рисуется некая грубая оболочка объекта, часто - AABB (axis-aligned bounding box), на неё удобно накладывать 3D-текстуру
  - Можно с **front-face culling**, чтобы алгоритм работал даже в случае, если камера находится внутри объекта
- Во фрагментный шейдер передаются координаты точки объекта; по ней вычисляется луч из камеры в точку объекта: $c + d \cdot t$, где c - позиция камеры, d - единичный вектор направления

# Volume rendering: raymarching

- Рисуется некая грубая оболочка объекта, часто - AABB (axis-aligned bounding box), на неё удобно накладывать 3D-текстуру
  - Можно с **front-face culling**, чтобы алгоритм работал даже в случае, если камера находится внутри объекта
- Во фрагментный шейдер передаются координаты точки объекта; по ней вычисляется луч из камеры в точку объекта: $c + d \cdot t$, где c - позиция камеры, d - единичный вектор направления
- Вычисляется пересечение луча с AABB - отрезок значений $[t_{min}, t_{max}]$, для которых соответствующие точки луча лежат внутри AABB
  - Если $t_{min} < 0$, нужно сделать его равным 0, иначе в рендеринг попадут точки, находящиеся сзади камеры

# Volume rendering: raymarching

- Выбирается количество шагов N, т.е. количество частей, на которые будет разбит отрезок $[t_{min}, t_{max}]$: фиксированное, или в зависимости от длины отрезка

# Volume rendering: raymarching

- Выбирается количество шагов N, т.е. количество частей, на которые будет разбит отрезок $[t_{min}, t_{max}]$: фиксированное, или в зависимости от длины отрезка
- Вдоль отрезка вычисляется аппроксимация интеграла $\int_0^L \exp\left(-\int_x^L k_a(s)ds\right) k_e(x)dx$ в виде $\sum_{i=0}^{N-1} \exp\left(-\sum_{j=i}^{N-1} k_a(x_j)\Delta x\right) k_e(x_i)\Delta x$

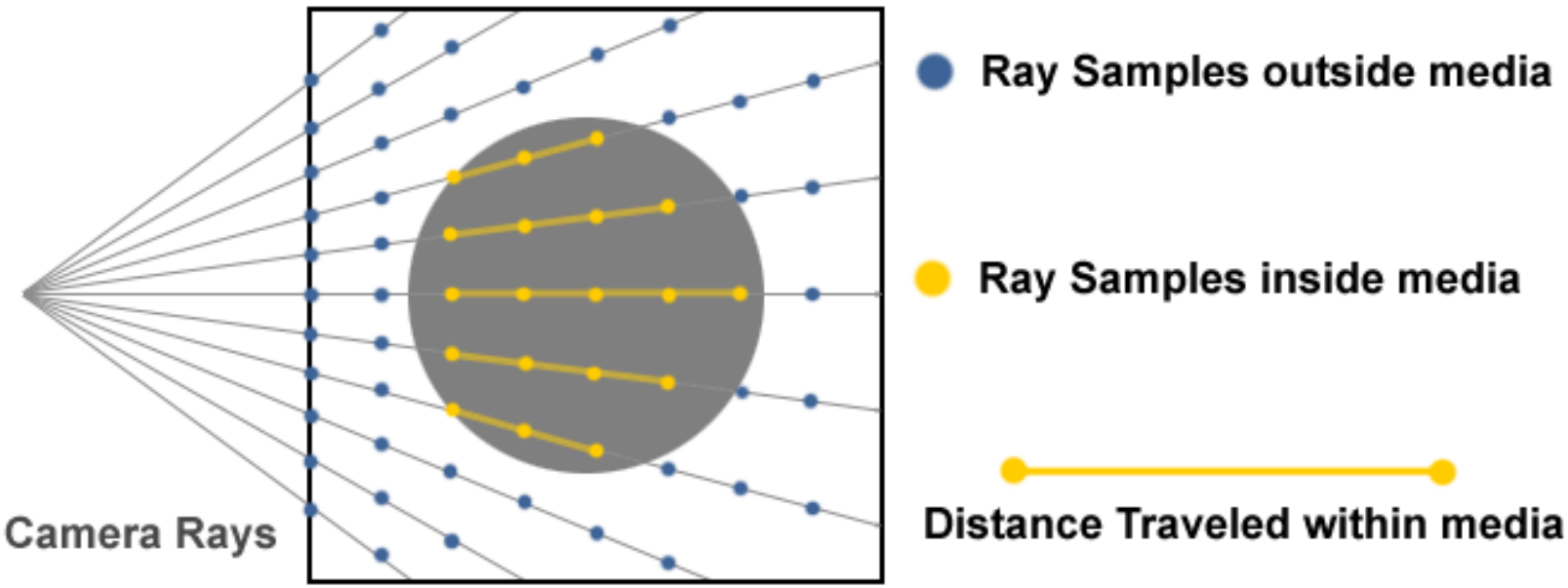
# Volume rendering: raymarching

- Выбирается количество шагов N, т.е. количество частей, на которые будет разбит отрезок $[t_{min}, t_{max}]$: фиксированное, или в зависимости от длины отрезка
- Вдоль отрезка вычисляется аппроксимация интеграла $\int_0^L \exp\left(-\int_x^L k_a(s)ds\right) k_e(x)dx$ в виде $\sum_{i=0}^{N-1} \exp\left(-\sum_{j=i}^{N-1} k_a(x_j)\Delta x\right) k_e(x_i)\Delta x$
- В качестве $x_i$ лучше брать центр i-ого отрезка

# Volume rendering: raymarching

- Выбирается количество шагов N, т.е. количество частей, на которые будет разбит отрезок $[t_{min}, t_{max}]$: фиксированное, или в зависимости от длины отрезка
- Вдоль отрезка вычисляется аппроксимация интеграла $\int_0^L \exp\left(-\int_x^L k_a(s)ds\right) k_e(x)dx$ в виде $\sum_{i=0}^{N-1} \exp\left(-\sum_{j=i}^{N-1} k_a(x_j)\Delta x\right) k_e(x_i)\Delta x$
- В качестве $x_i$ лучше брать центр i-ого отрезка
- Важно следить за направлением: интеграл выводился для луча идущего **в направлении камеры**, а интегрирование мы можем делать как **в направлении камеры** (back-to-front, от $t_{max}$ к $t_{min}$), так и **в направлении от камеры** (back-to-front, от $t_{min}$ к $t_{max}$)

# Volume rendering: raymarching



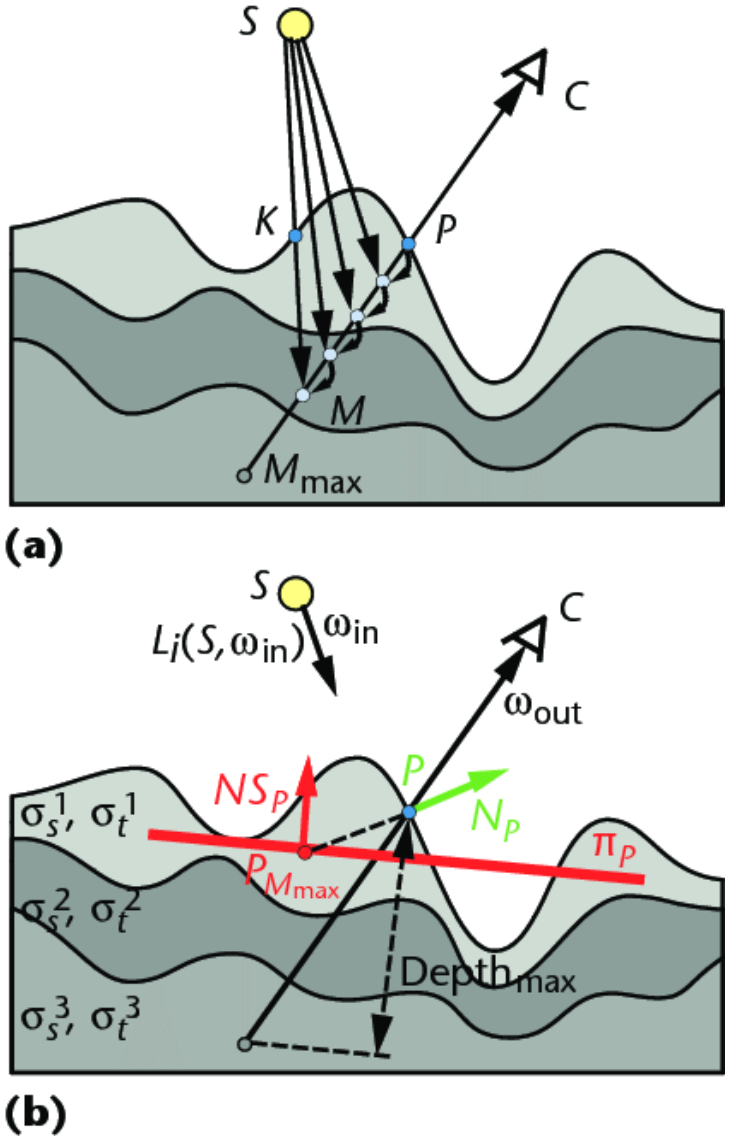
# Volume rendering: raymarching

- Для учёта рассеяния/освещения можно посылать дополнительные лучи, обрабатываемые тем же алгоритмом

# Volume rendering: raymarching

- Для учёта рассеяния/освещения можно посылать дополнительные лучи, обрабатываемые тем же алгоритмом
- Например, послать луч в сторону источника света, вычислить вдоль него optical depth, и по ней получить значение освещённости в точке (в GLSL это будет цикл внутри внешнего цикла)

# Volume rendering: raymarching



# Volume rendering: оптимизации

- Для хорошего качества нужно довольно много шагов вдоль луча (десятки, сотни)

# Volume rendering: оптимизации

- Для хорошего качества нужно довольно много шагов вдоль луча (десятки, сотни)
- С учётом внутреннего цикла для рассеяния/освещённости получается ещё на порядок больше

# Volume rendering: оптимизации

- Для хорошего качества нужно довольно много шагов вдоль луча (десятки, сотни)
- С учётом внутреннего цикла для рассеяния/освещённости получается ещё на порядок больше
- Работает очень медленно

# Volume rendering: оптимизации

- Для хорошего качества нужно довольно много шагов вдоль луча (десятки, сотни)
- С учётом внутреннего цикла для рассеяния/освещённости получается ещё на порядок больше
- Работает очень медленно
- Типичные оптимизации:
  - Использовать mipmap'ы 3D текстуры для хранения максимальной (вместо усреднённой) полупрозрачности; модифицировать алгоритм, чтобы он пропускал большие полностью прозрачные куски

# Volume rendering: оптимизации

- Для хорошего качества нужно довольно много шагов вдоль луча (десятки, сотни)
- С учётом внутреннего цикла для рассеяния/освещённости получается ещё на порядок больше
- Работает очень медленно
- Типичные оптимизации:
  - Использовать mipmap'ы 3D текстуры для хранения максимальной (вместо усреднённой) полупрозрачности; модифицировать алгоритм, чтобы он пропускал большие полностью прозрачные куски
  - Заканчивать алгоритм при front-to-back проходе, когда текущий вычисленный цвет уже имеет непрозрачность выше некого порогового значения (напр. 0.99)

# Volume rendering: оптимизации

- Для хорошего качества нужно довольно много шагов вдоль луча (десятки, сотни)
- С учётом внутреннего цикла для рассеяния/освещённости получается ещё на порядок больше
- Работает очень медленно
- Типичные оптимизации:
  - Использовать mipmap'ы 3D текстуры для хранения максимальной (вместо усреднённой) полупрозрачности; модифицировать алгоритм, чтобы он пропускал большие полностью прозрачные куски
  - Заканчивать алгоритм при front-to-back проходе, когда текущий вычисленный цвет уже имеет непрозрачность выше некого порогового значения (напр. 0.99)
  - Рисовать объект в меньшем разрешении

# Volume rendering: ссылки

- GPU Gems, Chapter 39. Volume Rendering Techniques
- GPU Gems 2, Chapter 16. Accurate Atmospheric Scattering
- Туториал по slicing (старый OpenGL!)
- Подробнее про цвет неба
- A Scalable and Production Ready Sky and Atmosphere Rendering Technique (статья 2020 года)
- Видео-туториал про рендеринг атмосферы