

Компьютерная графика

Лекция 12: Anti-aliasing, multisampling, volume rendering equation, scattering, volume slicing, raymarching

2023

Aliasing

- Термин из обработки сигналов: артефакт, возникающий при восстановлении сигнала по недостаточно большому количеству измерений

Aliasing

- Термин из обработки сигналов: артефакт, возникающий при восстановлении сигнала по недостаточно большому количеству измерений
- В графике: ‘лесенка’ из пикселей на границе объекта



Anti-aliasing: методы

- Вручную сглаживать градиентом (2D, карты, UI)

Anti-aliasing: методы

- Вручную сглаживать градиентом (2D, карты, UI)
- Supersampling (SSAA): рисовать увеличенную (x4) картинку в текстуру, затем уменьшать размер до требуемого, усредняя значения пикселей
 - Работает приемлемо, но очень дорого

Anti-aliasing: методы

- Вручную сглаживать градиентом (2D, карты, UI)
- Supersampling (SSAA): рисовать увеличенную (x4) картинку в текстуру, затем уменьшать размер до требуемого, усредняя значения пикселей
 - Работает приемлемо, но очень дорого
- Multisampling (MSAA)

Anti-aliasing: методы

- Вручную сглаживать градиентом (2D, карты, UI)
- Supersampling (SSAA): рисовать увеличенную (x4) картинку в текстуру, затем уменьшать размер до требуемого, усредняя значения пикселей
 - Работает приемлемо, но очень дорого
- Multisampling (MSAA)
- Пост-обработка (FXAA, SMAA, TAA)

Multisampling

- Алиасинг обычно возникает на границе треугольников (проблемы внутри решаются тірттар'ами текстур)

Multisampling

- Алиасинг обычно возникает на границе треугольников (проблемы внутри решаются тірттар'ами текстур)
- Выберем набор случайных точек (семплов) внутри пикселя (обычно от 2 до 16), будем хранить в каждом пикселе цвет каждой точки

Multisampling

- Алиасинг обычно возникает на границе треугольников (проблемы внутри решаются тірттар'ами текстур)
- Выберем набор случайных точек (семплов) внутри пикселя (обычно от 2 до 16), будем хранить в каждом пикселе цвет каждой точки
- Запустим фрагментный шейдер **один раз на пиксель**

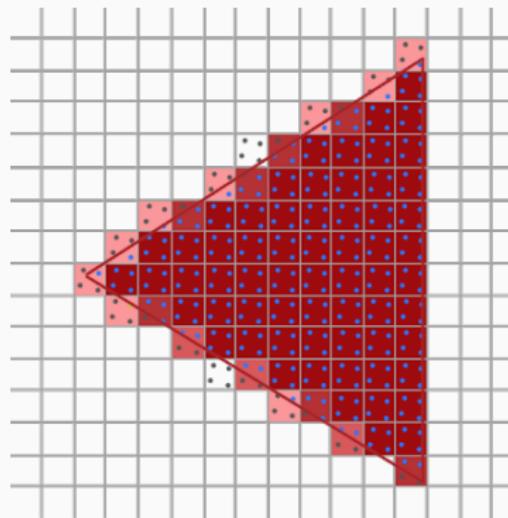
Multisampling

- Алиасинг обычно возникает на границе треугольников (проблемы внутри решаются тірттар'ами текстур)
- Выберем набор случайных точек (семплов) внутри пикселя (обычно от 2 до 16), будем хранить в каждом пикселе цвет каждой точки
- Запустим фрагментный шейдер **один раз на пиксель**
- Запишем результат вызова шейдера только в те точки, которые покрывает треугольник

Multisampling

- Алиасинг обычно возникает на границе треугольников (проблемы внутри решаются тірттар'ами текстур)
- Выберем набор случайных точек (семплов) внутри пикселя (обычно от 2 до 16), будем хранить в каждом пикселе цвет каждой точки
- Запустим фрагментный шейдер **один раз на пиксель**
- Запишем результат вызова шейдера только в те точки, которые покрывает треугольник
- При выводе на экран усредним значения в точках

Multisampling



Multisampling

- — Требует много памяти, большая нагрузка на шины данных
- + Фрагментный шейдер вызывается только один раз на пиксель

Multisampling в OpenGL

- Текстура с target = GL_TEXTURE_2D_MULTISAMPLE (данные инициализируются через glTexImage2DMultisample, позволяет задать количество семплов)

Multisampling в OpenGL

- Текстура с target = GL_TEXTURE_2D_MULTISAMPLE (данные инициализируются через glTexImage2DMultisample, позволяет задать количество семплов)
 - Нет mipmap'ов и фильтрации, читать из шейдера можно только через glTexelFetch (позволяет прочитать конкретный семпл)

Multisampling в OpenGL

- Текстура с target = GL_TEXTURE_2D_MULTISAMPLE (данные инициализируются через glTexImage2DMultisample, позволяет задать количество семплов)
 - Нет mipmap'ов и фильтрации, читать из шейдера можно только через glTexelFetch (позволяет прочитать конкретный семпл)
- Renderbuffer: glRenderbufferStorageMultisample

Multisampling в OpenGL

- Текстура с target = GL_TEXTURE_2D_MULTISAMPLE (данные инициализируются через glTexImage2DMultisample, позволяет задать количество семплов)
 - Нет mipmap'ов и фильтрации, читать из шейдера можно только через glTexelFetch (позволяет прочитать конкретный семпл)
- Renderbuffer: glRenderbufferStorageMultisample
- Рисование во фреймбуфер с такими текстурами/renderbuffer'ами будет автоматически делать multisampling

Multisampling в OpenGL

- Текстура с target = GL_TEXTURE_2D_MULTISAMPLE (данные инициализируются через glTexImage2DMultisample, позволяет задать количество семплов)
 - Нет mipmap'ов и фильтрации, читать из шейдера можно только через glTexelFetch (позволяет прочитать конкретный семпл)
- Renderbuffer: glRenderbufferStorageMultisample
- Рисование во фреймбуфер с такими текстурами/renderbuffer'ами будет автоматически делать multisampling
- Вывести multisampled текстуру/renderbuffer на экран (*multisample resolve*) можно с помощью glBindFramebuffer

FXAA, SMAA, TAA

- FXAA (Fast approXimate Anti-Aliasing): специальный шейдер пост-обработки, на основе эвристик вычисляющий места, в которых нужно сгладить изображение
 - Недорогой алгоритм, приемлемо работает

FXAA, SMAA, TAA

- FXAA (Fast approXimate Anti-Aliasing): специальный шейдер пост-обработки, на основе эвристик вычисляющий места, в которых нужно сгладить изображение
 - Недорогой алгоритм, приемлемо работает
- SMAA (Subpixel Morphological Anti-Aliasing): как FXAA, но больше эвристик
 - Дороже чем FXAA, результат лучше

FXAA, SMAA, TAA

- FXAA (Fast approXimate Anti-Aliasing): специальный шейдер пост-обработки, на основе эвристик вычисляющий места, в которых нужно сгладить изображение
 - Недорогой алгоритм, приемлемо работает
- SMAA (Subpixel Morphological Anti-Aliasing): как FXAA, но больше эвристик
 - Дороже чем FXAA, результат лучше
- TAA (Temporal Anti-Aliasing): усредняем значение пикселя со значением этого же пикселя на предыдущем кадре с учётом изменившегося положения камеры (*temporal reprojection*)
 - Нужны эвристики чтобы отличить тот же пиксель с предыдущего кадра от случайно туда попавшего пикселя
 - Нужно знать скорость движения пикселей для движущихся объектов, чтобы правильно работал temporal reprojection
 - Приводит к лёгкому размытию при движении (из-за чего его часто ругают)
 - Обычно используется в крупных движках

Anti-aliasing: ссылки

- khronos.org/opengl/wiki/Multisampling
- learnopengl.com/Advanced-OpenGL/Anti-Aliasing
- Статья про разные методы антиалиасинга

Volume rendering

- Рендеринг объектов, заданных распределением свойств в пространстве:

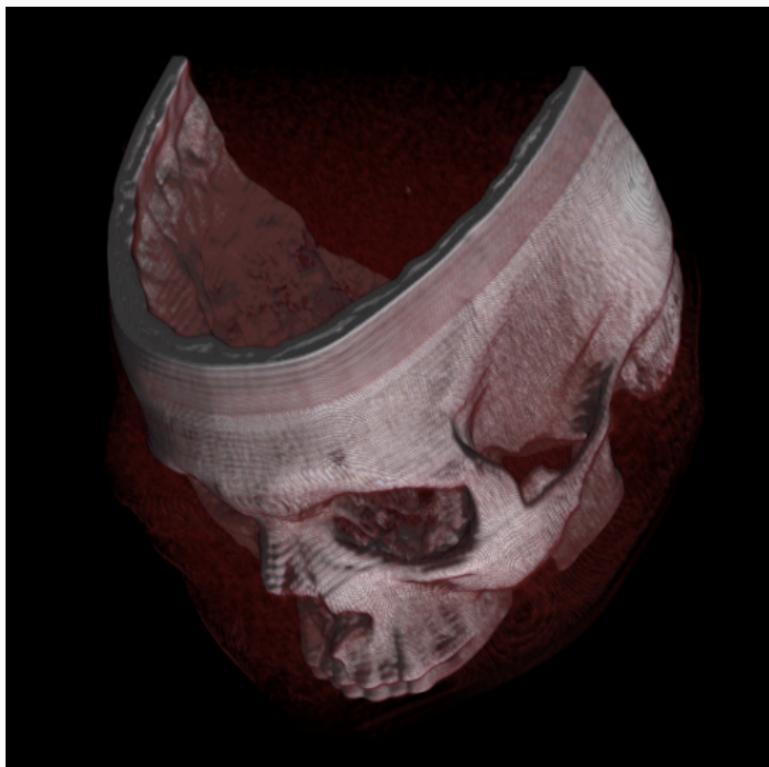
Volume rendering

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

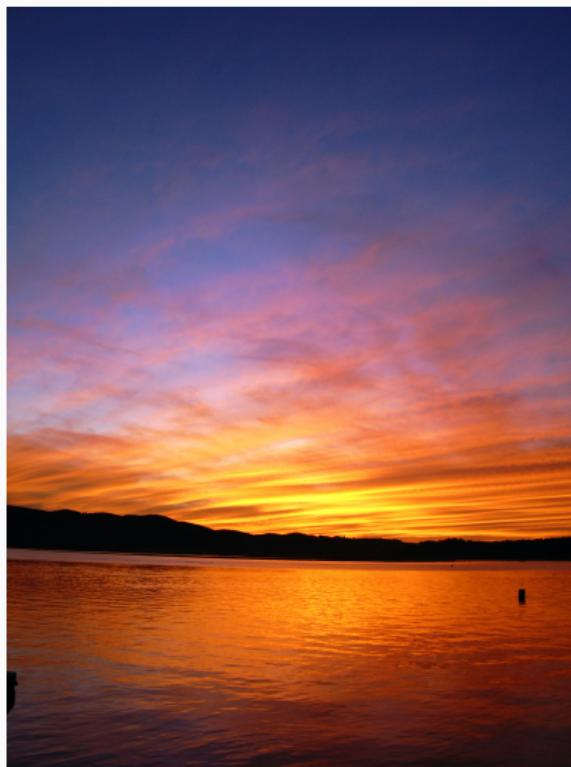
Volume rendering

- Рендеринг объектов, заданных распределением свойств в пространстве:
 - Цвет
 - Прозрачность
 - Параметры рассеивания
- Медицина: КТ-сканы, МРТ-сканы
- Физика: атмосфера, космические объекты (туманности), жидкости и газы
- Аппроксимации, основанные на теории volume рендеринга: дым, туман, облака, небо, subsurface scattering

Volume-rendering черепа



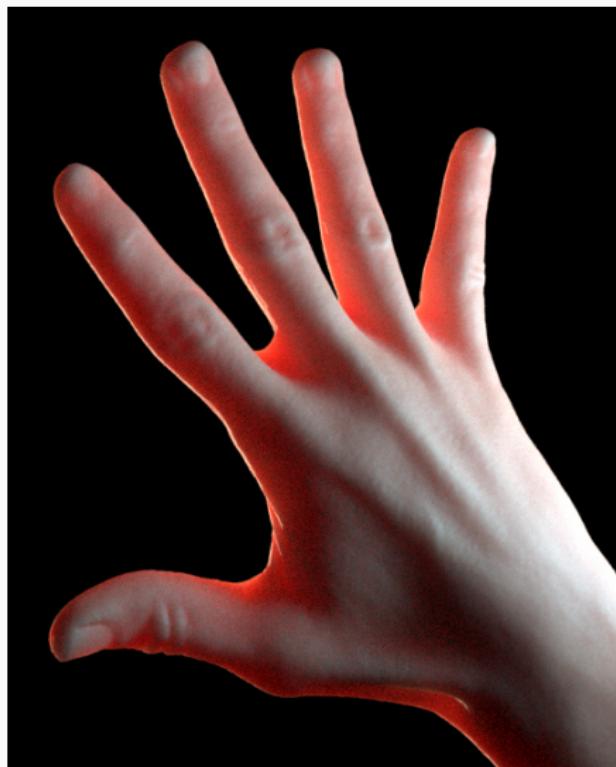
He6o (Nishita model)



ДЫМ



Subsurface scattering



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*)
- $\text{absorption} + \text{scattering} = \text{extinction}$

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*)
- $\text{absorption} + \text{scattering} = \text{extinction}$
- Распространение света в такой среде описывается интегро-дифференциальным уравнением для точки (как меняется свет, идущий через точку в таком-то направлении) либо интегральным уравнением для отрезка (как меняется свет, идущий из одной точки в другую точку)

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
 - Направления движения света

Volume rendering: теория

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

Volume rendering: теория

- Коэффициенты поглощения k_a , рассеяния k_s и излучения k_e могут зависеть от
 - Точки пространства
 - Направления движения света
 - Длины волны
 - Угла (в случае рассеяния)

Volume rendering: теория

- Коэффициенты поглощения k_a , рассеяния k_s и излучения k_e могут зависеть от
 - Точки пространства
 - Направления движения света
 - Длины волны
 - Угла (в случае рассеяния)
- 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
 - Направления движения света
 - Длины волны
 - Угла (в случае рассеяния)
- 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· м^{-1}

Volume rendering: теория

- Коэффициенты поглощения k_a , рассеяния k_s и излучения k_e могут зависеть от
 - Точки пространства
 - Направления движения света
 - Длины волны
 - Угла (в случае рассеяния)
- 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 $\text{люкс}\cdot\text{м}^{-1}$
 - N.B.: часто как $k_e = k_a + k_s$ обозначают коэффициент исчезновения (extinction)

Volume rendering: теория

- Коэффициенты поглощения k_a , рассеяния k_s и излучения k_e могут зависеть от
 - Точки пространства
 - Направления движения света
 - Длины волны
 - Угла (в случае рассеяния)
- 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· м^{-1}
 - N.B.: часто как $k_e = k_a + k_s$ обозначают коэффициент исчезновения (extinction)
- Эти параметры могут также задаваться на единицу плотности вещества (тогда при использовании их надо домножить на плотность)

Volume rendering: absorption (Beer–Lambert law)

- Пусть есть среда с постоянным коэффициентом поглощения k_a

Volume rendering: absorption (Beer–Lambert law)

- Пусть есть среда с постоянным коэффициентом поглощения k_a
 - Вдоль бесконечно малого расстояния Δx поглотится $k_a \Delta x$ от общего количестве проходящего света

Volume rendering: absorption (Beer–Lambert law)

- Пусть есть среда с постоянным коэффициентом поглощения k_a
 - Вдоль бесконечно малого расстояния Δx поглотится $k_a \Delta x$ от общего количестве проходящего света
- Как много света $I(p, \omega)$, выходящего из некой точки p в направлении ω , поглотится на отрезке длины L , т.е. на отрезке $[p, p + L\omega]$?

Volume rendering: absorption (Beer–Lambert law)

- Пусть есть среда с постоянным коэффициентом поглощения k_a
 - Вдоль бесконечно малого расстояния Δx поглотится $k_a \Delta x$ от общего количестве проходящего света
- Как много света $I(p, \omega)$, выходящего из некой точки p в направлении ω , поглотится на отрезке длины L , т.е. на отрезке $[p, p + L\omega]$?
- Разобьём отрезок на N частей равной длины $\Delta x = \frac{L}{N}$

Volume rendering: absorption (Beer–Lambert law)

- Пусть есть среда с постоянным коэффициентом поглощения k_a
 - Вдоль бесконечно малого расстояния Δx поглотится $k_a \Delta x$ от общего количестве проходящего света
- Как много света $I(p, \omega)$, выходящего из некой точки p в направлении ω , поглотится на отрезке длины L , т.е. на отрезке $[p, p + L\omega]$?
- 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 $(1 - k_a \frac{L}{N})^2$

Volume rendering: absorption (Beer–Lambert law)

- Пусть есть среда с постоянным коэффициентом поглощения k_a
 - Вдоль бесконечно малого расстояния Δx поглотится $k_a \Delta x$ от общего количестве проходящего света
- Как много света $I(p, \omega)$, выходящего из некой точки p в направлении ω , поглотится на отрезке длины L , т.е. на отрезке $[p, p + L\omega]$?
- 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 $(1 - k_a \frac{L}{N})^2$
- Всего после N отрезков останется $(1 - k_a \frac{L}{N})^N$ света

Volume rendering: absorption (Beer–Lambert law)

- Пусть есть среда с постоянным коэффициентом поглощения k_a
 - Вдоль бесконечно малого расстояния Δx поглотится $k_a \Delta x$ от общего количестве проходящего света
- Как много света $I(p, \omega)$, выходящего из некой точки p в направлении ω , поглотится на отрезке длины L , т.е. на отрезке $[p, p + L\omega]$?
- 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 $(1 - k_a \frac{L}{N})^2$
- Всего после N отрезков останется $(1 - k_a \frac{L}{N})^N$ света
- В пределе: $\lim_{N \rightarrow \infty} (1 - k_a \frac{L}{N})^N = \exp(-k_a L)$

Volume rendering: absorption

- В пределе: $I(p + L\omega, \omega) = \exp(-k_a L)I(p, \omega)$

Volume rendering: absorption

- В пределе: $I(p + L\omega, \omega) = \exp(-k_a L)I(p, \omega)$
 - 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 $L \rightarrow \infty$ и $k_a > 0$, то весь свет поглотится

Volume rendering: absorption

- В пределе: $I(p + L\omega, \omega) = \exp(-k_a L)I(p, \omega)$
 - 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 $L \rightarrow \infty$ и $k_a > 0$, то весь свет поглотится
- Если k_a не постоянна, получим $\exp\left(-\int_0^L k_a(p + t\omega)dt\right)$

Volume rendering: absorption

- В пределе: $I(p + L\omega, \omega) = \exp(-k_a L)I(p, \omega)$
 - 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 $L \rightarrow \infty$ и $k_a > 0$, то весь свет поглотится
- Если k_a не постоянна, получим $\exp\left(-\int_0^L k_a(p + t\omega)dt\right)$
- К тому же результату можно прийти, написав дифференциальное уравнение:
$$\frac{d}{dt}I(p + t\omega, \omega) = -k_a(p + t\omega)I(p + t\omega, \omega)$$

Volume rendering: absorption

- В пределе: $I(p + L\omega, \omega) = \exp(-k_a L)I(p, \omega)$
 - 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 $L \rightarrow \infty$ и $k_a > 0$, то весь свет поглотится
- Если k_a не постоянна, получим $\exp\left(-\int_0^L k_a(p + t\omega)dt\right)$
- К тому же результату можно прийти, написав дифференциальное уравнение:
$$\frac{d}{dt}I(p + t\omega, \omega) = -k_a(p + t\omega)I(p + t\omega, \omega)$$
- Величина $\int_0^L k_a(p + t\omega)dt$ называется оптической глубиной (*optical depth*)

Volume rendering: emission

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

Volume rendering: emission

- Пусть есть среда с постоянным коэффициентом излучения k_e , т.е. для бесконечно малого расстояния Δx излучится $k_e \Delta x$ света
- Как много света излучится на отрезке длины $[p, p + L\omega]$ в направлении ω ?

Volume rendering: emission

- Пусть есть среда с постоянным коэффициентом излучения k_e , т.е. для бесконечно малого расстояния Δx излучится $k_e \Delta x$ света
- Как много света излучится на отрезке длины $[p, p + L\omega]$ в направлении ω ?
- $I(p + L\omega, \omega) = I(p, \omega) + k_e L$

Volume rendering: emission

- Пусть есть среда с постоянным коэффициентом излучения k_e , т.е. для бесконечно малого расстояния Δx излучится $k_e \Delta x$ света
- Как много света излучится на отрезке длины $[p, p + L\omega]$ в направлении ω ?
- $I(p + L\omega, \omega) = I(p, \omega) + k_e L$
- Если k_e не постоянна: $I(p + L\omega, \omega) = I(p, \omega) + \int_0^L k_e(p + t\omega) dt$

Volume rendering: absorption + emission

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

Volume rendering: absorption + emission

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

Volume rendering: absorption + emission

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

Volume rendering: absorption + emission

- Пусть есть среда с постоянным коэффициентом поглощения k_a и излучения k_e , т.е. для бесконечно малого расстояния Δx излучится $k_e \Delta x$ и поглотится $k_a \Delta x$ 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: absorption + emission

- Пусть есть среда с постоянным коэффициентом поглощения k_a и излучения k_e , т.е. для бесконечно малого расстояния Δx излучится $k_e \Delta x$ и поглотится $k_a \Delta x$ 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
$$(1 + (1 - k_a \Delta x) + (1 - k_a \Delta x)^2) k_e \Delta x$$

Volume rendering: absorption + emission

- Пусть есть среда с постоянным коэффициентом поглощения k_a и излучения k_e , т.е. для бесконечно малого расстояния Δx излучится $k_e \Delta x$ и поглотится $k_a \Delta x$ 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
$$(1 + (1 - k_a \Delta x) + (1 - k_a \Delta x)^2) k_e \Delta x$$
- 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: absorption + emission

- Пусть есть среда с постоянным коэффициентом поглощения k_a и излучения k_e , т.е. для бесконечно малого расстояния Δx излучится $k_e \Delta x$ и поглотится $k_a \Delta x$ 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
$$(1 + (1 - k_a \Delta x) + (1 - k_a \Delta x)^2) k_e \Delta x$$
- 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:
$$\lim_{N \rightarrow \infty} \frac{1 - (1 - k_a \Delta x)^N}{k_a} k_e = \frac{1 - \exp(-k_a L)}{k_a} k_e$$

Volume rendering: absorption + emission

- $I(p + L\omega, \omega) = \frac{1 - \exp(-k_a L)}{k_a} k_e$

Volume rendering: absorption + emission

- $I(p + L\omega, \omega) = \frac{1 - \exp(-k_a L)}{k_a} k_e$
- Сингулярность при $k_a = 0$, в пределе $k_a \rightarrow 0$ получим Lk_e

Volume rendering: absorption + emission

- $I(p + L\omega, \omega) = \frac{1 - \exp(-k_a L)}{k_a} k_e$
- Сингулярность при $k_a = 0$, в пределе $k_a \rightarrow 0$ получим Lk_e
- В пределе при $L \rightarrow \infty$ получим $\frac{k_e}{k_a}$

Volume rendering: absorption + emission

- $I(p + L\omega, \omega) = \frac{1 - \exp(-k_a L)}{k_a} k_e$
- Сингулярность при $k_a = 0$, в пределе $k_a \rightarrow 0$ получим Lk_e
- В пределе при $L \rightarrow \infty$ получим $\frac{k_e}{k_a}$
- Если k_a и k_e не постоянны, получим

$$I(p + L\omega, \omega) = \int_0^L k_e(p + t\omega) \exp\left(-\int_t^L k_a(p + s\omega) ds\right) dt$$

Volume rendering: absorption + emission

- $I(p + L\omega, \omega) = \frac{1 - \exp(-k_a L)}{k_a} k_e$
- Сингулярность при $k_a = 0$, в пределе $k_a \rightarrow 0$ получим Lk_e
- В пределе при $L \rightarrow \infty$ получим $\frac{k_e}{k_a}$
- Если k_a и k_e не постоянны, получим

$$I(p + L\omega, \omega) = \int_0^L k_e(p + t\omega) \exp\left(-\int_t^L k_a(p + s\omega) ds\right) dt$$

- Соответствующее дифференциальное уравнение:
$$\frac{d}{dt} I(p + t\omega, \omega) = k_e(p + t\omega) - k_a(p + t\omega)I(p + t\omega, \omega)$$

Volume rendering: absorption + emission

- Если $I(p, \omega) \neq 0$, то

$$I(p + L\omega, \omega) = \exp(-k_a L) I(p, \omega) + (1 - \exp(-k_a L)) \frac{k_e}{k_a}$$

Volume rendering: absorption + emission

- Если $I(p, \omega) \neq 0$, то

$$I(p + L\omega, \omega) = \exp(-k_a L) I(p, \omega) + (1 - \exp(-k_a L)) \frac{k_e}{k_a}$$

- Если коэффициенты не постоянны, то

$$\begin{aligned} I(p + L\omega, \omega) &= I(p, \omega) \exp \left(- \int_0^L k_a(p + t\omega) dt \right) + \\ &+ \int_0^L k_e(p + t\omega) \exp \left(- \int_t^L k_a(p + s\omega) ds \right) dt \quad (1) \end{aligned}$$

Volume rendering: absorption + emission

- $\exp(-k_a L)I(p, \omega) + (1 - \exp(-k_a L))\frac{k_e}{k_a}$ можно интерпретировать как линейную интерполяцию между $I(p, \omega)$ и $\frac{k_e}{k_a}$ с коэффициентом $1 - \exp(-k_a L)$

Volume rendering: absorption + emission

- $\exp(-k_a L)l(p, \omega) + (1 - \exp(-k_a L))\frac{k_e}{k_a}$ можно интерпретировать как линейную интерполяцию между $l(p, \omega)$ и $\frac{k_e}{k_a}$ с коэффициентом $1 - \exp(-k_a L)$
- Формализует полупрозрачность:
 - $l(p, \omega)$ – свет, приходящий сзади объекта в сторону камеры
 - k_e/k_a – цвет (излучение) самого объекта
 - $\exp(-k_a L)$ – непрозрачность (opacity)
 - $1 - \exp(-k_a L)$ – прозрачность (transparency)

Volume rendering: absorption + emission

- $\exp(-k_a L)l(p, \omega) + (1 - \exp(-k_a L))\frac{k_e}{k_a}$ можно интерпретировать как линейную интерполяцию между $l(p, \omega)$ и $\frac{k_e}{k_a}$ с коэффициентом $1 - \exp(-k_a L)$
- Формализует полупрозрачность:
 - $l(p, \omega)$ – свет, приходящий сзади объекта в сторону камеры
 - k_e/k_a – цвет (излучение) самого объекта
 - $\exp(-k_a L)$ – непрозрачность (opacity)
 - $1 - \exp(-k_a L)$ – прозрачность (transparency)
- Формализует туман:
 - k_e/k_a – цвет тумана
 - k_a – коэффициент ‘густоты’ тумана (чем больше, тем сильнее туман)

Туман: код шейдера

```
vec3 color = ...;
vec3 fog_color = ...;
float fog_absorption = ...;

float camera_distance = length(camera_position - position);
float optical_depth = camera_distance * fog_absorption;
float fog_factor = 1 - exp(-optical_depth);
color = mix(color, fog_color, fog_factor);
```

Volume rendering: рассеяние (scattering)

- Любая среда не только поглощает и излучает свет, но и *рассеивает* его, т.е. меняет направление его движения

Volume rendering: рассеяние (scattering)

- Любая среда не только поглощает и излучает свет, но и *рассеивает* его, т.е. меняет направление его движения
- Рассеяние описывается фазовой функцией (phase function):
 $f(p, \theta)$ – количество света, рассеянное на угол θ в точке p

Volume rendering: рассеяние (scattering)

- Любая среда не только поглощает и излучает свет, но и *рассеивает* его, т.е. меняет направление его движения
- Рассеяние описывается фазовой функцией (phase function):
 $f(p, \theta)$ – количество света, рассеянное на угол θ в точке p
- Аналог BRDF для volume rendering'a

Volume rendering: рассеяние (scattering)

- Любая среда не только поглощает и излучает свет, но и *рассеивает* его, т.е. меняет направление его движения
- Рассеяние описывается фазовой функцией (phase function):
 $f(p, \theta)$ – количество света, рассеянное на угол θ в точке p
- Аналог BRDF для volume rendering'a
- Вид функции зависит от конкретных составляющих среды, часто 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 N₂ и O₂)
- $f \cong \frac{1+\cos(\theta)^2}{\lambda^4}$

Рассеяние Релея (Rayleigh)

- Рассеяние света частицами намного меньшими длины волны (например, молекулами газов N₂ и O₂)
- $f \cong \frac{1+\cos(\theta)^2}{\lambda^4}$
- Больше рассеяния вперёд и назад, чем в стороны

Рассеяние Релея (Rayleigh)

- Рассеяние света частицами намного меньшими длины волны (например, молекулами газов N₂ и O₂)
- $f \cong \frac{1+\cos(\theta)^2}{\lambda^4}$
- 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 N₂ и O₂)
- $f \cong \frac{1+\cos(\theta)^2}{\lambda^4}$
- 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:

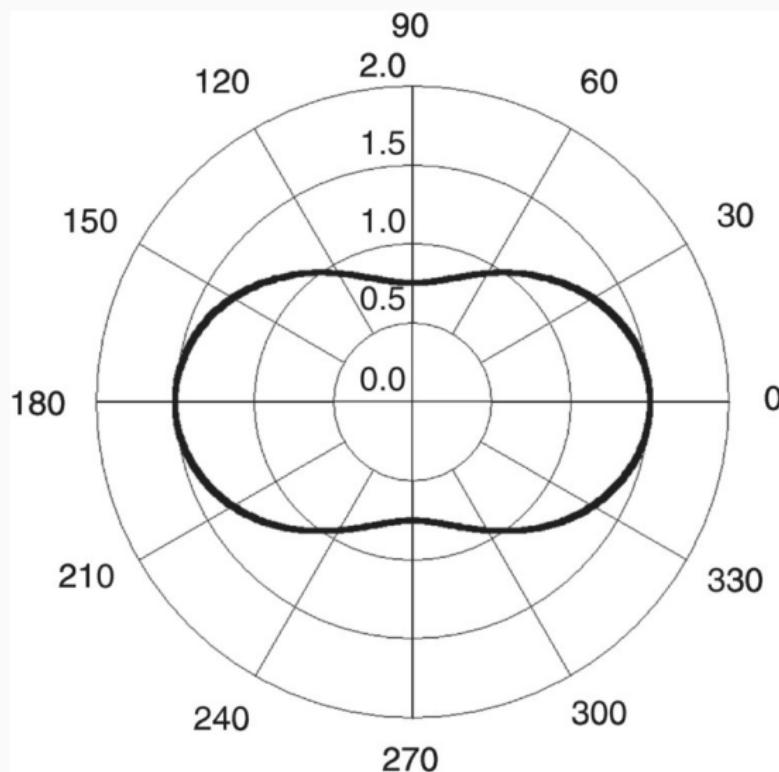
Рассеяние Релея (Rayleigh)

- Рассеяние света частицами намного меньшими длины волны (например, молекулами газов N₂ и O₂)
- $f \cong \frac{1+\cos(\theta)^2}{\lambda^4}$
- 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:
 - Свет, проходящий сквозь атмосферу, рассеивается в основном в синем диапазоне, поэтому небо синее

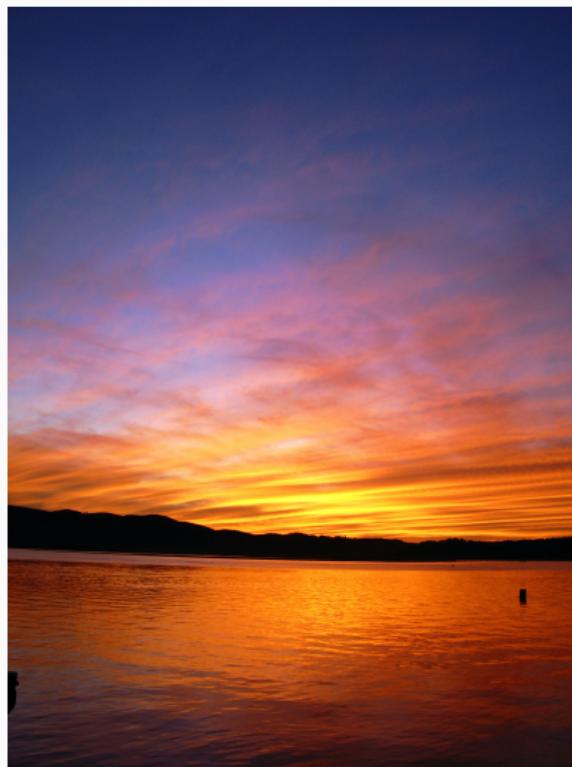
Рассеяние Релея (Rayleigh)

- Рассеяние света частицами намного меньшими длины волны (например, молекулами газов N₂ и O₂)
- $f \cong \frac{1+\cos(\theta)^2}{\lambda^4}$
- 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:
 - Свет, проходящий сквозь атмосферу, рассеивается в основном в синем диапазоне, поэтому небо синее
 - Свет, идущий от солнца на закате/восходе, дальше идёт через атмосферу и теряет весь синий свет и часть зелёного, поэтому выглядит оранжевым или красным

Рассеяние Релея: phase function



Рассеяние Релея: цвет неба



Рассеяние Мие (Mie)

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

Рассеяние Мie (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

Henyey-Greenstein phase function (1941)

$$f(\theta) = \frac{1}{4\pi} \frac{1 - g^2}{(1 + g^2 - 2g \cos \theta)^{3/2}} \quad (2)$$

- g – настраиваемый параметр (средний косинус угла рассеяния)
- Часто используется как аппроксимация вместо более сложных функций

Volume rendering: рассеяние

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

Volume rendering: рассеяние

- Рассеяние значительно усложняет уравнения: свет, пришедший из любого направления, может рассеяться в сторону камеры в любой точке
- При рассчёте рассеяния 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
- Интерполяция по сетке из кубов/тетраэдров/etc

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 функции, 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 функции, задающей объект
- Слои накладываются друг на друга с blending’ом

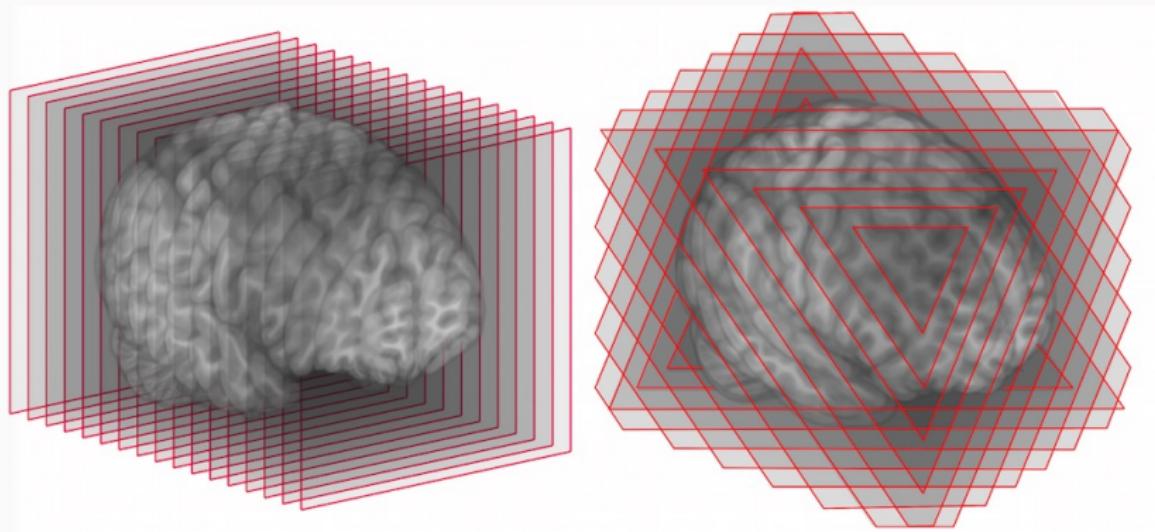
Volume rendering: slicing

- 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 функции, 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 функции, 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'e, нужно отсортировать частицы, правильно настроить blending и значение альфа-канала

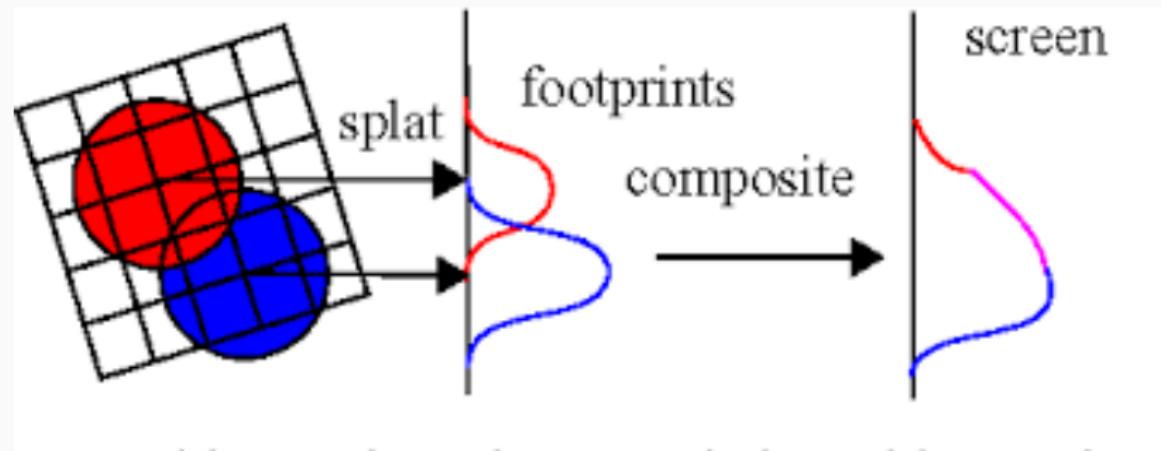
Volume rendering: splatting

- 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'e, 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'e, 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



Gaussian splatting

- Gaussian splatting – популярный в последнее время метод рендеринга 3D сцен, полученных по набору изображений с разных ракурсов (напр. фотографий)

Gaussian splatting

- Gaussian splatting – популярный в последнее время метод рендеринга 3D сцен, полученных по набору изображений с разных ракурсов (напр. фотографий)
- Сцена представляется набором частиц-гауссиан (*gaussian splats*)

Gaussian splatting

- Gaussian splatting – популярный в последнее время метод рендеринга 3D сцен, полученных по набору изображений с разных ракурсов (напр. фотографий)
- Сцена представляется набором частиц-гауссиан (*gaussian splats*)
- В этом методе цвет точки на экране – явная (хоть и довольно сложная) функция от параметров всех частиц

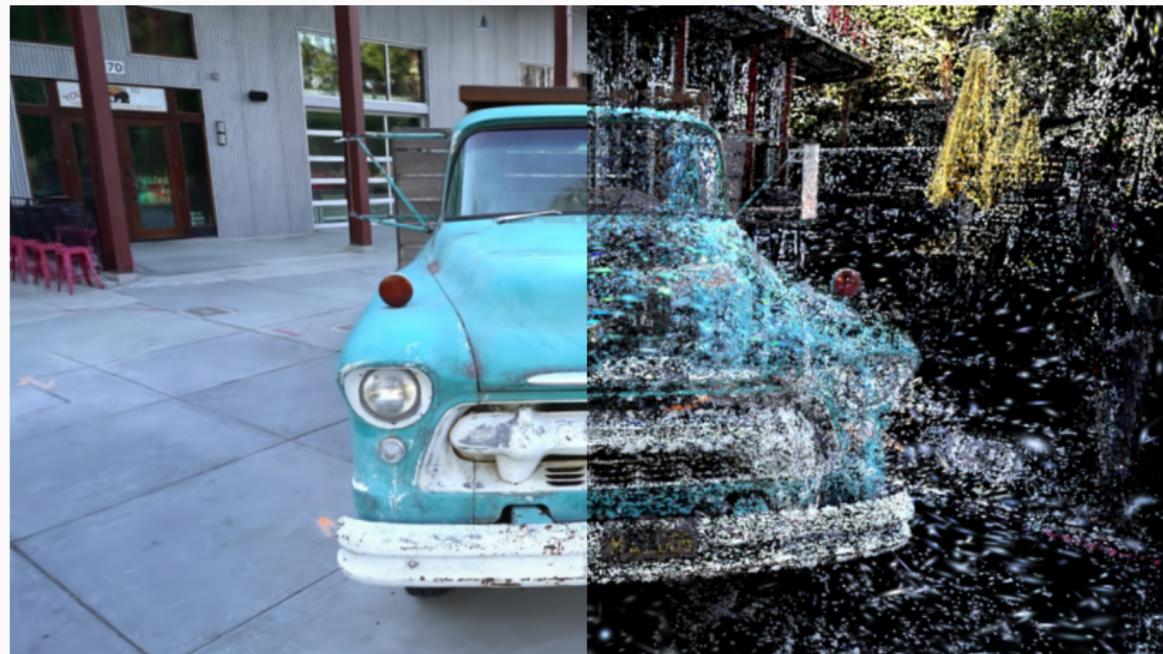
Gaussian splatting

- Gaussian splatting – популярный в последнее время метод рендеринга 3D сцен, полученных по набору изображений с разных ракурсов (напр. фотографий)
- Сцена представляется набором частиц-гауссиан (*gaussian splats*)
- В этом методе цвет точки на экране – явная (хоть и довольно сложная) функция от параметров всех частиц
- \Rightarrow можно оптимизировать параметры частиц, чтобы при рендеринге они давали исходные фотографии

Gaussian splatting

- Gaussian splatting – популярный в последнее время метод рендеринга 3D сцен, полученных по набору изображений с разных ракурсов (напр. фотографий)
- Сцена представляется набором частиц-гауссиан (*gaussian splats*)
- В этом методе цвет точки на экране – явная (хоть и довольно сложная) функция от параметров всех частиц
- \Rightarrow можно оптимизировать параметры частиц, чтобы при рендеринге они давали исходные фотографии
- Результат оптимизации – вычисленный набор параметров частиц, описывающий нашу сцену

Gaussian splatting



Ray*

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

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

Raycasting



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

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 фотorealистичного 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

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

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 частей N разбиения отрезка $[t_{min}, t_{max}]$ (фиксированное, или в зависимости от длины отрезка)

Volume rendering: raymarching

- Выбирается количество частей N разбиения отрезка $[t_{min}, t_{max}]$ (фиксированное, или в зависимости от длины отрезка)
- Вдоль отрезка вычисляется аппроксимация интеграла
$$\int_0^L \exp\left(-\int_x^L k_a(s)ds\right) k_e(t)dt$$
 в виде
$$\sum_{i=0}^{N-1} \exp\left(-\sum_{j=i}^{N-1} k_a(t_j)\Delta t\right) k_e(t_i)\Delta t$$

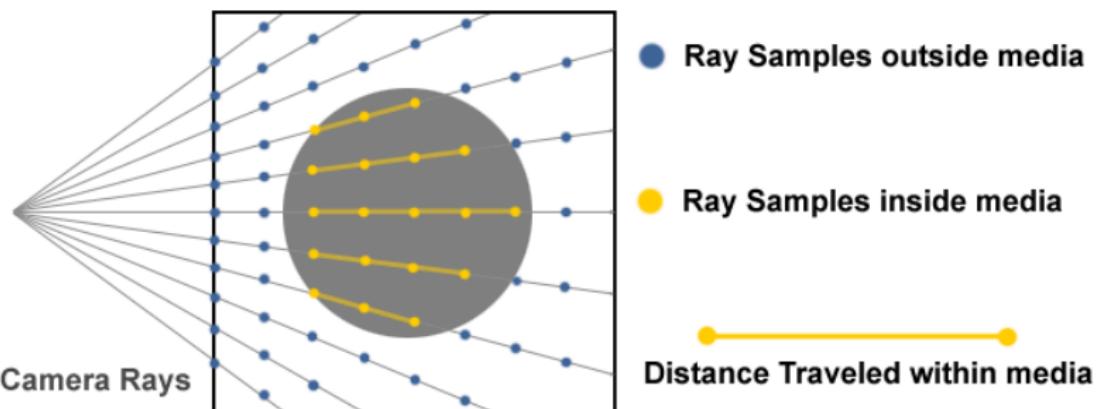
Volume rendering: raymarching

- Выбирается количество частей N разбиения отрезка $[t_{min}, t_{max}]$ (фиксированное, или в зависимости от длины отрезка)
- Вдоль отрезка вычисляется аппроксимация интеграла
$$\int_0^L \exp\left(-\int_x^L k_a(s)ds\right) k_e(t)dt$$
 в виде
$$\sum_{i=0}^{N-1} \exp\left(-\sum_{j=i}^{N-1} k_a(t_j)\Delta t\right) k_e(t_i)\Delta t$$
- В качестве t_i лучше брать центр i -ого отрезка

Volume rendering: raymarching

- Выбирается количество частей N разбиения отрезка $[t_{min}, t_{max}]$ (фиксированное, или в зависимости от длины отрезка)
- Вдоль отрезка вычисляется аппроксимация интеграла
$$\int_0^L \exp\left(-\int_x^L k_a(s)ds\right) k_e(t)dt$$
в виде
$$\sum_{i=0}^{N-1} \exp\left(-\sum_{j=i}^{N-1} k_a(t_j)\Delta t\right) k_e(t_i)\Delta t$$
- В качестве t_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** (front-to-back, от t_{min} к t_{max})

Volume rendering: raymarching



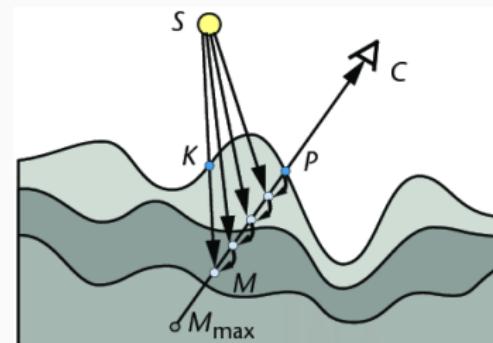
Volume rendering: raymarching

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

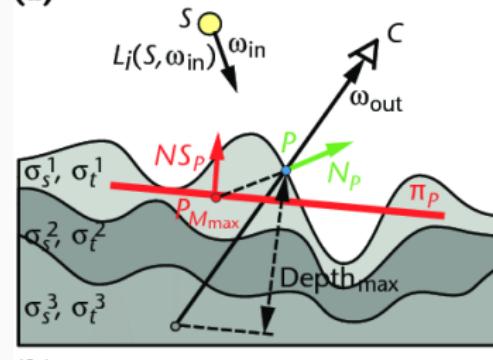
Volume rendering: raymarching

- Для учёта рассеяния/освещения можно посыпать 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 как количество света, которое не поглотилось и не рассеялось (в GLSL это будет цикл внутри внешнего цикла)

Volume rendering: raymarching



(a)



(b)

Raymarching (только absorption): алгоритм

```
dt = (t_max - t_min) / N
opt_depth = 0
for i in 0..N-1:
    t = t_min + (i + 0.5) * dt
    p = start + t * direction
    opt_depth += k_a(p) * dt
abs_factor = exp(-opt_depth)
color = back_color * abs_factor
```

Raymarching (только absorption): альтернативный алгоритм

```
dt = (t_max - t_min) / N
color = back_color
for i in 0..N-1:
    t = t_min + (i + 0.5) * dt
    p = start + t * direction
    color *= exp(- k_a(p) * dt)
```

Raymarching (absorption+emission): front-to-back алгоритм

```
dt = (t_max - t_min) / N
opt_depth = 0
color = back_color
for i in 0..N-1:
    t = t_min + (i + 0.5) * dt
    p = start + t * direction
    opt_depth += k_a(p) * dt
    color += exp(-opt_depth) * k_e(p) * dt
```

Raymarching (absorption+emission): альтернативный front-to-back алгоритм

```
dt = (t_max - t_min) / N
color = back_color
for i in 0..N-1:
    t = t_min + (i + 0.5) * dt
    p = start + t * direction
    color += k_e(p) * dt
    color *= exp(- k_a(p) * dt)
```

Raymarching (+single scattering в направлении света): front-to-back алгоритм

```
dt = (t_max - t_min) / N
color = back_color
for i in 0..N-1:
    t = t_min + (i + 0.5) * dt
    p = start + t * direction

    light_opt_depth = 0
    // [s_min .. s_max] - пересечение луча
    // в направлении света с AABB объекта
    ds = (s_max - s_min) / M
    for j in 0..M-1:
        s = s_min + (j + 0.5) * ds
        q = p + s * light_direction
        light_opt_depth += (k_a(q) + k_s(q)) * ds

    color += light_color * exp(-light_opt_depth)
        * k_s(p) * dt
        * phase_f(p, light_direction, direction)

    color += k_e(p) * dt
    color *= exp(-(k_a(p) + k_s(p)) * dt)
```

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 тіртар'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 тіртар'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 тіртар'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