Компьютерная графика

Финальный проект

2024

Задание

- Можно сделать любое количество заданий из описанных в данных слайдах

Задание

- Можно сделать любое количество заданий из описанных в данных слайдах
- Каждое задание – отдельный мини-проект с некоторым количеством самостоятельного изучения

Задание

- Можно сделать любое количество заданий из описанных в данных слайдах
- Каждое задание – отдельный мини-проект с некоторым количеством самостоятельного изучения
- Как обычно, можно сделать задание не полностью

Задание

- Можно сделать любое количество заданий из описанных в данных слайдах
- Каждое задание – отдельный мини-проект с некоторым количеством самостоятельного изучения
- Как обычно, можно сделать задание не полностью
- Во всех заданиях должны двигаться камера и источники света (если они есть), если не написано обратное

Задание

- Можно сделать любое количество заданий из описанных в данных слайдах
- Каждое задание – отдельный мини-проект с некоторым количеством самостоятельного изучения
- Как обычно, можно сделать задание не полностью
- Во всех заданиях должны двигаться камера и источники света (если они есть), если не написано обратное
- Все альбедо-текстуры должны загружаться как sRGB

Задание

- Можно сделать любое количество заданий из описанных в данных слайдах
- Каждое задание – отдельный мини-проект с некоторым количеством самостоятельного изучения
- Как обычно, можно сделать задание не полностью
- Во всех заданиях должны двигаться камера и источники света (если они есть), если не написано обратное
- Все альбедо-текстуры должны загружаться как sRGB
- Во всех заданиях должны быть гамма-коррекция и какой-нибудь tone mapping (лучше Uncharted или ACES)

Что можно использовать

- Можно пользоваться вспомогательными библиотеками (например, для загрузки текстур, моделей и сцен)

Что можно использовать

- Можно пользоваться вспомогательными библиотеками (например, для загрузки текстур, моделей и сцен)
- Весь OpenGL-код должен быть написан вами, т.е. библиотека не должна загружать данные на GPU, создавать текстуры, и т.п.

Что можно использовать

- Можно пользоваться вспомогательными библиотеками (например, для загрузки текстур, моделей и сцен)
- Весь OpenGL-код должен быть написан вами, т.е. библиотека не должна загружать данные на GPU, создавать текстуры, и т.п.
- Можно брать код из практик и домашних заданий, в т.ч. загрузчики моделей, анимаций и шрифтов (при необходимости их можно доработать)

Что можно использовать

- Можно пользоваться вспомогательными библиотеками (например, для загрузки текстур, моделей и сцен)
- Весь OpenGL-код должен быть написан вами, т.е. библиотека не должна загружать данные на GPU, создавать текстуры, и т.п.
- Можно брать код из практик и домашних заданий, в т.ч. загрузчики моделей, анимаций и шрифтов (при необходимости их можно доработать)
- Можно брать сцены и текстуры из практик и домашних заданий

Список заданий

- Планета Земля с volumetric атмосферой
- Большой лес с impostor-деревьями
- Metaballs
- Бассейн с водой

Планета

Планета: задание

- Сфера с текстурой нашей планеты
- Освещение по Фонгу движущимся солнцем
- На неосвещённой стороне используем другую текстуру “ночной Земли”
- Карта glossines для specular-освещения (это будет, в основном, вода)
- Карта высот планеты, сдвигающая вершины вверх
  - Лучше высоты искуственно на что-нибудь домножить, чтобы горы было лучше видно
  - Здесь, возможно, придётся добавить геометрический шейдер для расчёта нормалей
- Тени обычным shadow-mapping’ом
- Volumetric слой атмосферы (сфера чуть большего радиуса вокруг Земли) с рассеянием и объёмными тенями
- Облака – ещё одна сфера где-то посередине между уровнем моря и самыми высокими горами
  - На них тоже должно быть освещение (можно только диффузное) и тени
  - Они сами должны отбрасывать тень на Землю

Планета: советы

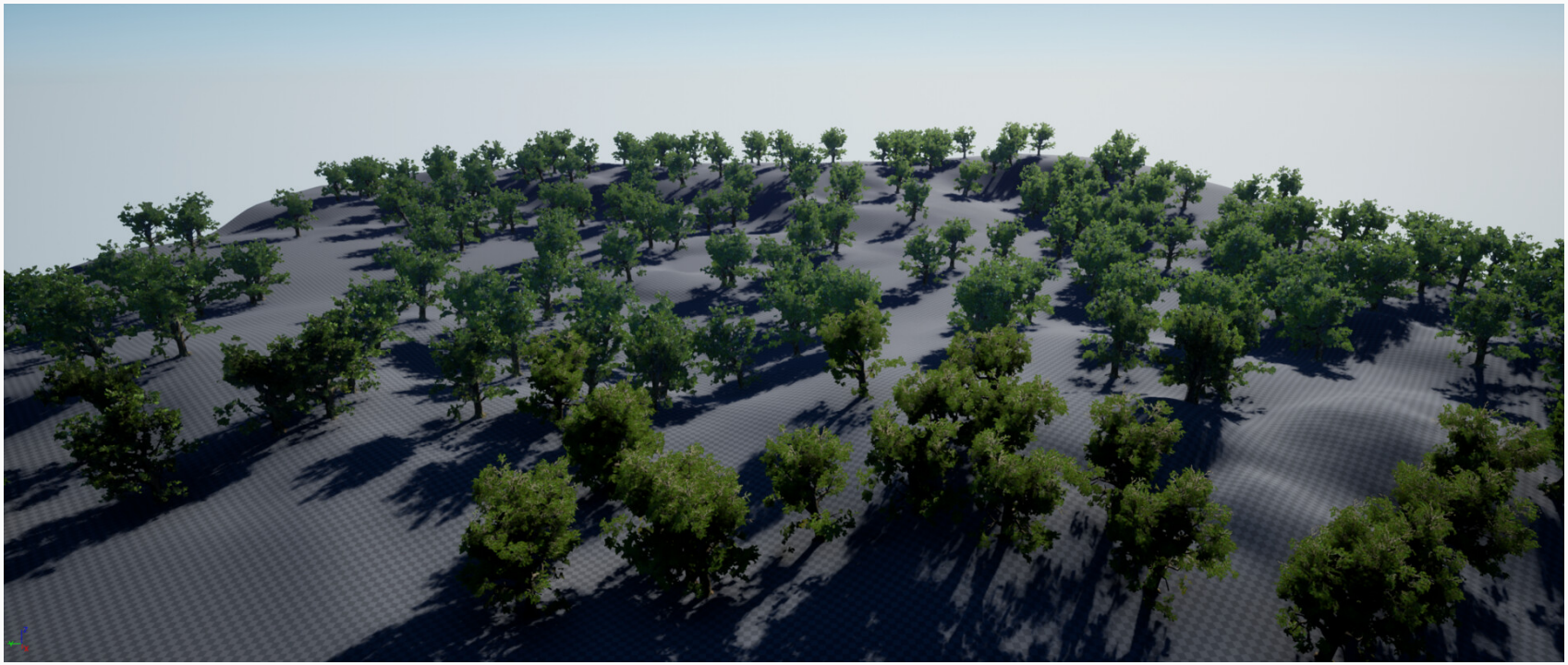
- Вершины сферы можно сгенерировать самим, проще всего через сферические координаты, равномерным шагом по широте и долготе
- Можно не хранить сами вершины, и вычислять их из номера вершины в шейдере
- Все текстуры легко гуглятся, главное – чтобы они использовали географическую проекцию и не были обрезаны по полюсам
- В карте высот надо разобраться, в каком формате хранятся высоты
- В карте высот могут быть глубины океанов, их лучше заменить нулём (уровнем моря)
- В вычислении проекции для shadow mapping'а нужно учесть, что вы поднимаете точки сферы на значение высоты
- Накладывать атмосферу можно в шейдере, рисующем Землю/облака, но тогда нужно нарисовать ещё одну сферу с front-face culling, чтобы нарисовалась часть атмосферы в космосе
- Можно рисовать атмосферу отдельной сферой (с обычным back-face culling) вокруг планеты со своим шейдером (и с прозрачностью!), но тогда вам нужен свой буфер глубины, чтобы прочитать из него расстояние до нарисованной точки на поверхности/облаках
- Коэффициенты рассеяния Релея берём из реальных данных для красного, зелёного и синего цвета (тоже легко гуглятся)
  - Они зависят от высоты точки над поверхностью Земли, это важно учесть
  - Нужно быть внимательными с единицами измерения!
- Проще всего будет работать с реальными величинами: радиус Земли ≈ 6400 км, толщина атмосферы ≈ 100 км

Планета: баллы

- **1 балл**: есть сфера, освещённая по Фонгу
- **1 балл**: текстура альбедо Земли + ночной стороны Земли
- **1 балл**: текстура glossiness
- **1 балл**: вершины сдвигаются картой высот
- **2 балла**: по карте высот посчитаны нормали
- **2 балла**: shadow mapping
- **2 балла**: объёмная атмосфера с рассеянием
- **2 балла**: рассеяние учитывает тени
- **2 балла**: облака с тенями и отбрасывающие тень

Максимум: **14 баллов**

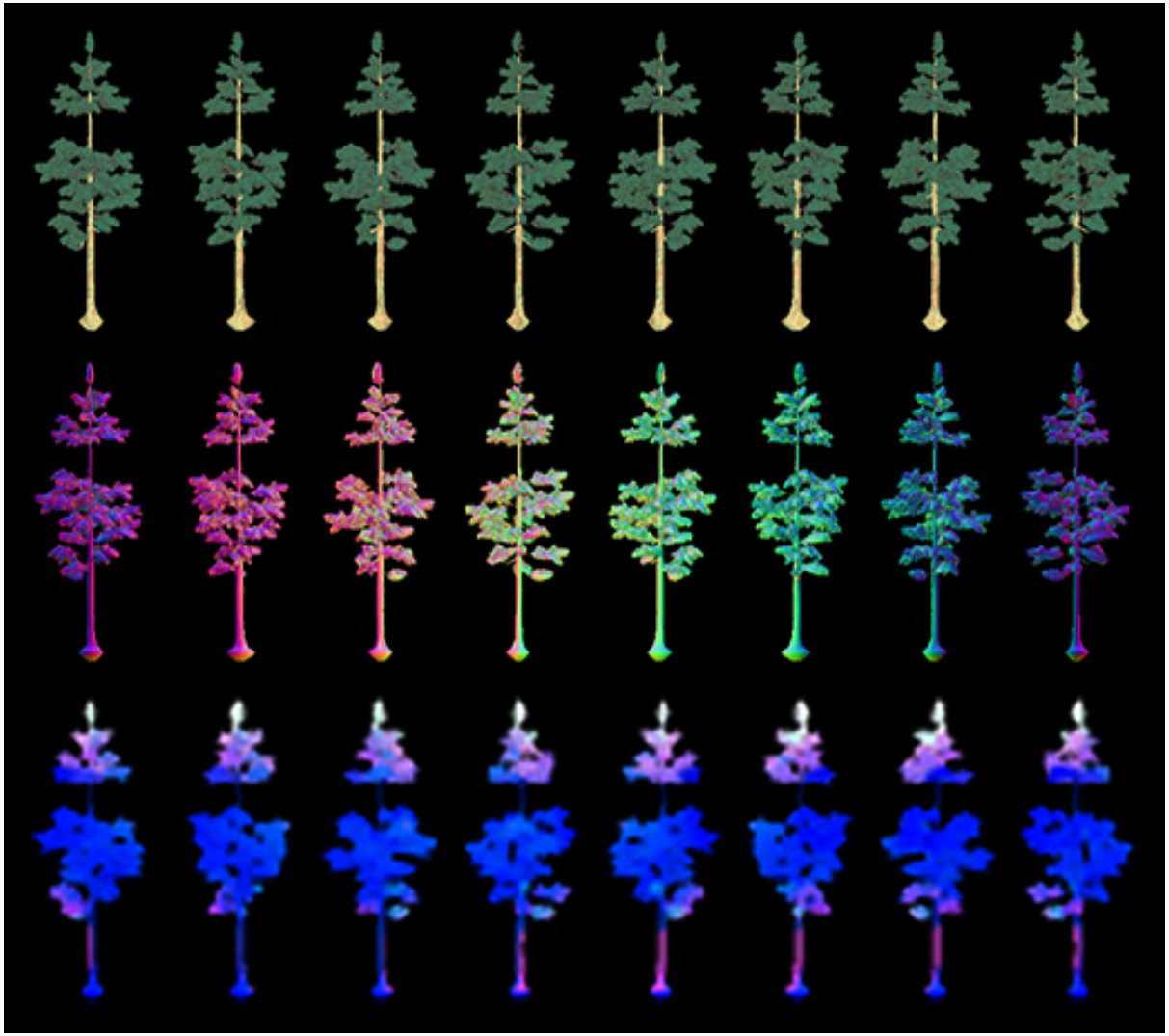
Лес



Лес: задание

- Плоскость с какой-нибудь текстурой (например, травы)
- На плоскости много (напр. 10000) деревьев, нарисованных техникой *impostors*:
  - Вместо объекта рисуется billboard – прямоугольник, повёрнутый в сторону камеры
  - На прямоугольник наносится одна из заранее подготовленных текстур объекта в зависимости от направления взгляда
- На старте программы по некой сетке направлений рендерим наше дерево (без освещения!) отдельно в текстуры альбедо, нормалей, материала, и глубины (а-ля gbuffer)
- Генерируем набор случайных позиций, поворотов (в горизонтальной плоскости), масштабов, и цветов деревьев, равномерно распределённых на плоскости
- В процессе работы программы instancing'ом или геометрическим шейдером рисуем все деревья одной командой рисования
- Конкретные текстуры для одного инстанса выбираются по направлению на камеру
- Вместо одного направления можно брать 4 соседних и интерполировать по ним
- На деревьях обычное освещение по Фонгу + тени (обычным shadow mapping'ом)

Лес: impostor текстуры



Лес: советы

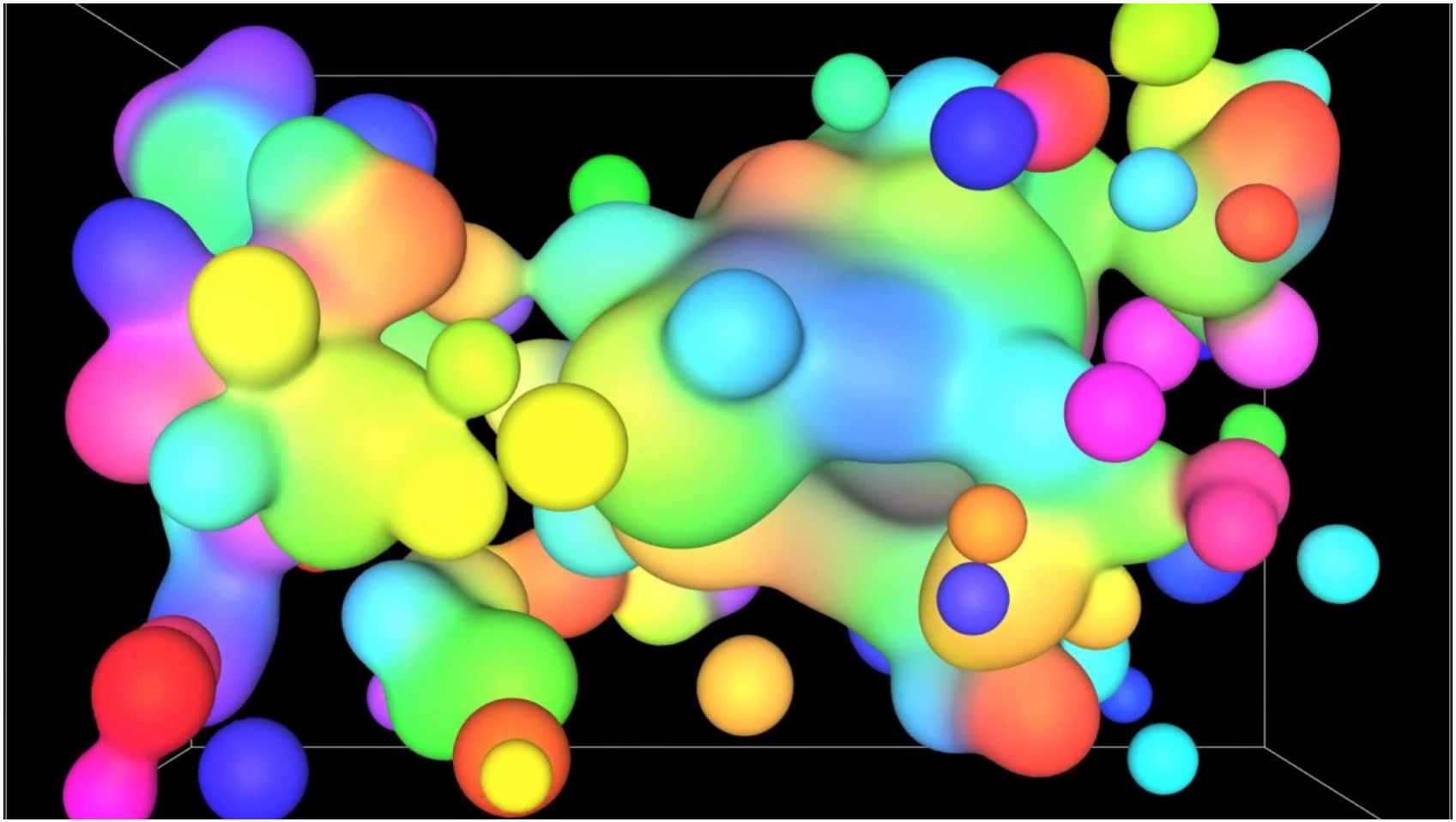
- В качестве объекта берём любую 3D-модель с деревом
- В качестве сетки направлений можно взять полусферу с фиксированными шагами по высоте и широте
- Для impostor текстур хорошо использовать array texture (один слой – одно направление из сетки)
- Вершинный/геометрический шейдер поворачивает прямоугольник в сторону камеры и выбирает номер слоя array-текстуры на основании направления на камеру и поворота самого инстанса дерева
- Фрагментный шейдер читает эти текстуры, вычисляет освещение, и выставляет `gl_FragDepth` используя буфер глубины impostor'а
- Цвет инстанса дерева можно просто смешать с текстурой альбедо перед вычислением освещения
- Не вся текстура impostor'а будет содержать дерево; для полупрозрачности можно использовать **`discard`**
- Текстура альбедо должна быть в sRGB!
- Нужно, чтобы корень дерева всегда оставался в одном и том же месте!
- Удобно расположить корень дерева (т.е. позицию инстанса) в фиксированном месте прямоугольника impostor'а: например, в центре по X, и в центре или в самом внизу по Y
- Для теней нужно использовать impostor, смотрящий в сторону источника света, а не в сторону камеры!

Лес: баллы

- **1 балл**: есть плоскость с текстурой, освещённая по Фонгу
- **2 балла**: инстансингом или геометрическим шейдером рисуются impostor-прямоугольники
- **1 балл**: у инстансов есть случайные повороты, масштабы, и цвета
- **2 балла**: генерируются и используются текстуры альбедо impostor'ов
- **2 балла**: генерируются и используются текстуры нормалей impostor'ов
- **1 балл**: генерируются и используются текстуры материала impostor'ов
- **1 балл**: генерируются и используются текстуры глубины impostor'ов
- **2 балла**: shadow mapping
- **2 балла**: интерполяция между 4 соседними направлениями для всех текстур impostor'ов

Максимум: **14 баллов**

Metaballs



Metaballs: задание

- Что-то в духе первого домашнего задания, но в 3D
- Описываем набор трёхмерных шариков, как-то движущихся в 3D, каждый со своей позицией p_i, радиусом r_i, и цветом c_i
- Рисуем изоповерхность функции $f(p) = K$ с настраиваемым значением K
- В качестве функции берём $f(p) = \sum w_i = \sum \exp\left(-\frac{|p-p_i|}{r_i}\right)$
- Цвета интерполируем с softmax-весами: итоговый цвет это $\sum \lambda_i c_i$ где $\lambda_i = \frac{w_i}{\sum w_j} = \frac{w_i}{f(p)}$
- Функцию и цвета нужно хранить как 3D текстуру и обновлять на каждый кадр (руками, или с помощью рендеринга по слоям, или compute shader'ом)
- Изоповерхность рисуем по квадратной сетке алгоритмом marching cubes или marching tetrahedra в геометрическом шейдере
- У поверхности должны быть нормали и освещение по Фонгу (со specular составляющей)

Metaballs: советы

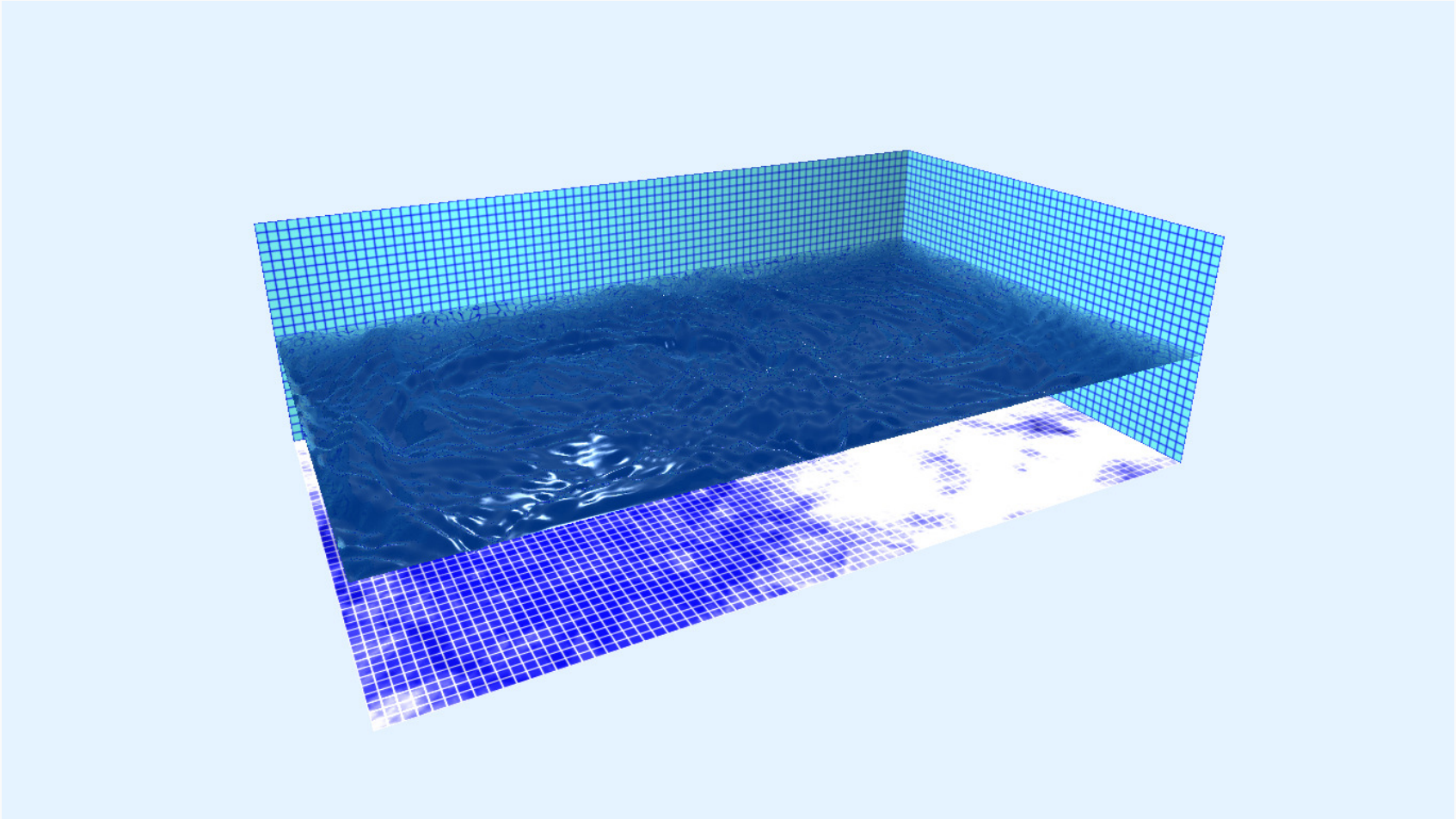
- Для генерации 3D текстуры с функцией на GPU можно передать все шарики в шейдер массивом uniform-переменных, uniform-блоком, или buffer texture
- Можно генерировать её аддитивным блендингом, так как функция это сумма вкладов каждого шара по отдельности (тогда не нужно передавать все шарики в шейдер – они рисуются по одному)
- В текстуре можно хранить цвет, умноженный на значение функции (т.е. $\sum w_i c_i$), и делить его на сумму весов уже при чтении этой текстуры, – тогда его тоже можно рисовать аддитивным блендингом
- Можно хранить и цвет, и значение функции в одной текстуре с форматом RGBA32F (RGB – цвет, A – значение функции)
- Кубическую сетку для marching cubes можно не хранить явно, а вычислять на основе координат вершин
- В marching cubes очень много случаев, вместо него можно разбить каждый квадрат сетки на тетраэдры (вдоль диагонали) и делать на них marching tetrahedra, в котором всего 4 различных случая
- Нормали можно взять плоские, а можно попробовать восстановить гладкие нормали по градиенту функции $\nabla f(p)$ (который можно посчитать конечными разностями)

Metaballs: баллы

- **1 балл**: как-нибудь симулируются летающие шарики
- **2 балла**: на CPU генерируется 3D текстура со значениями функции и цветами
- либо **3 балла**: текстура генерируется на GPU любым способом
  - **+2 балла**: текстура генерируется аддитивным блендингом
  - либо **+2 балла**: текстура генерируется compute shader'ом
- **3 балла**: строятся изоповерхности функции
- **2 балла**: изоповерхности раскрашены в правильные цвета
- **2 балла**: изоповерхности используют нормали и освещение по Фонгу
  - **+1 балл**: нормали гладкие, восстановлены по значениям функции

Максимум: **14 баллов**

Вода



Вода: задание

- Плоская поверхность “дна” с какой-нибудь простой повторяющейся текстурой и диффузным освещением от солнца (стенки “бассейна” рисовать не нужно)
- Динамическая поверхность воды на какой-то высоте над дном, высота и нормали вершин вычисляются на основе суммы каких-нибудь синусоид
- Вокруг – какой-нибудь environment map
- На воде должны быть отражения и преломления
- На дне должны быть каустики от солнца
- В этом задании солнце может быть статическим

Вода: советы

- Меш воды лучше взять довольно детализированный; координаты вершин можно вычислять на основе номера вершины
- Нормаль вычисляется явно через производную функции высоты $z = f(x, y)$ как $\left(-\frac{\partial f}{\partial x}, -\frac{\partial f}{\partial y}, 1\right)$
- Вода частично отражает, а частично преломляет свет; коэффициенты нужно взять из аппроксимации Шлика (Schlick) уравнений Френеля; для воды коэффициент преломления берём $n = 1.333$
- Угол преломлённого луча вычисляем через закон Снеллиуса (Snell's law)
- Для отражённого луча берём свет из environment map
- Преломлённый луч пересекаем руками с прямоугольником дна: если пересёкся, возвращаем цвет дна в этой точке (с учётом текстуры и освещения), иначе – тоже свет из environment map
- Каустики рисуем отдельным проходом в особую текстуру аддитивным блендингом: вершинный шейдер превращает каждую вершину в точку на дне, в которую попадёт преломлённый луч от солнца
- Наложенные каустики должны быть видны на самом дне, и через воду

Вода: баллы

- **1 балл**: есть плоскость дна с текстурой, освещённая по Фонгу
- **1 балл**: есть environment map
- **2 балла**: есть динамическая поверхность воды с плавными нормалями
- **2 балла**: есть отражение environment map
- **2 балла**: есть преломление
- **1 балла**: отражение + преломление по закону Френеля
- **3 балла**: генерируется текстура каустик
- **1 балл**: каустики видны на дне
- **1 балл**: каустики видны через воду

Максимум: **14 баллов**

Алгоритмы

- Любое задание, сделанное хотя бы на 5 баллов, можно выложить на GitHub, приложить красивый скриншот, и сделать Readme, в котором будет показан этот скриншот, – за это даётся ещё **1 балл**

Алгоритмы

- Любое задание, сделанное хотя бы на 5 баллов, можно выложить на GitHub, приложить красивый скриншот, и сделать Readme, в котором будет показан этот скриншот, – за это даётся ещё **1 балл**
- Если у вас есть другая идея, которую очень хочется реализовать – напишите мне, обсудим

Алгоритмы

- Любое задание, сделанное хотя бы на 5 баллов, можно выложить на GitHub, приложить красивый скриншот, и сделать Readme, в котором будет показан этот скриншот, – за это даётся ещё **1 балл**
- Если у вас есть другая идея, которую очень хочется реализовать – напишите мне, обсудим
- Сдача в день зачёта или экзамена