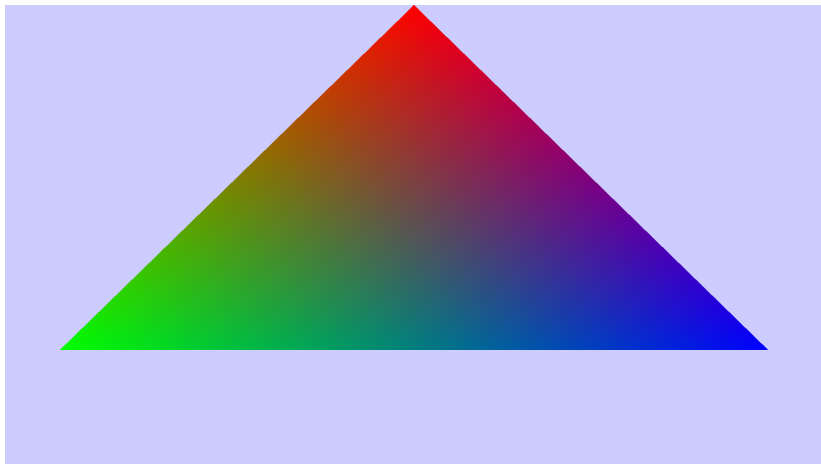


Компьютерная графика

Практика 2: Uniform'ы и матрицы преобразований

2021

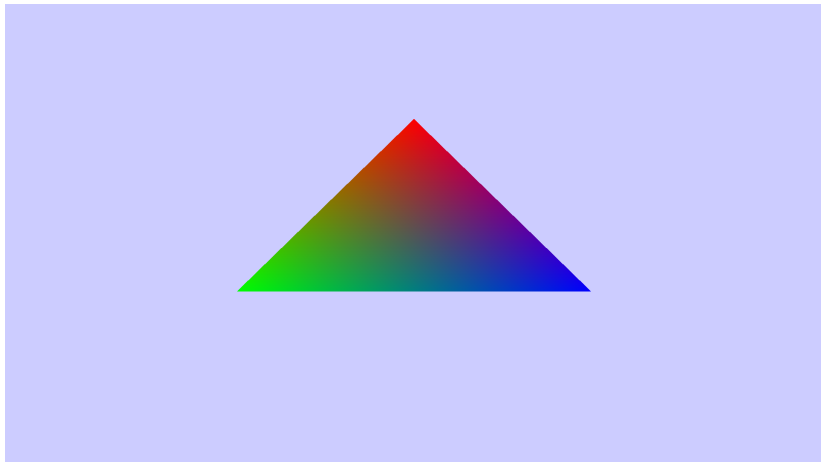


Задание 1

Уменьшим треугольник в 2 раза, используя uniform переменную

- ▶ В коде вершинного шейдера:
 - ▶ `uniform float scale;`
 - ▶ Нужно где-то умножить вектор координат на `scale`
 - ▶ NB: последняя координата `gl_Position` должна остаться равной 1
- ▶ После создания программы, до основного цикла:
 - ▶ `glUseProgram`
 - ▶ `glGetUniformLocation` – возвращает уникальный идентификатор, позволяющий работать с этой uniform-переменной
 - ▶ `glUniform1f` – устанавливает значение конкретной uniform-переменной типа `float`

Задание 1

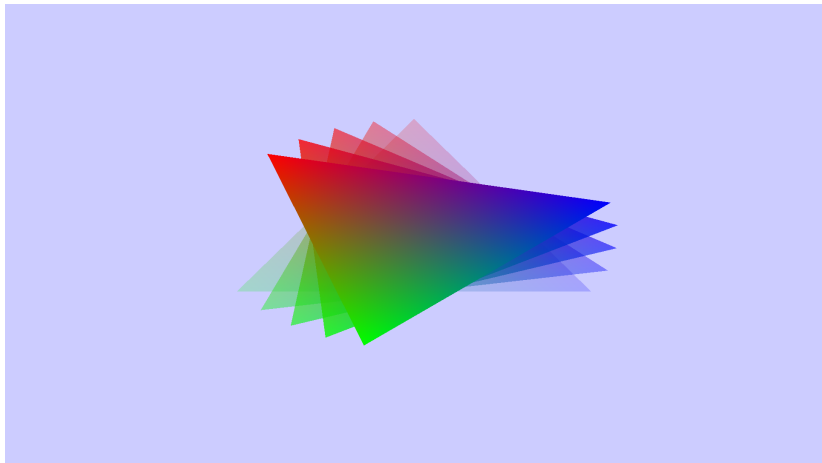


Задание 2

Заставим треугольник постоянно крутиться

- ▶ В коде вершинного шейдера:
 - ▶ `uniform float angle;`
 - ▶ Нужно где-то повернуть вектор координат на `angle`
 - ▶ Формула поворота есть в слайдах лекции (пока нужны только первые 2 координаты)
- ▶ После создания программы, до основного цикла:
 - ▶ `glUseProgram`
 - ▶ `glGetUniformLocation`
 - ▶ `float time = 0.f;`
- ▶ В теле основного цикла рендеринга, после вычисления `dt`
 - ▶ `time += dt;`
- ▶ В теле основного цикла рендеринга, после `glUseProgram`, до `glDrawArrays`
 - ▶ `glUniform1f`
 - ▶ В качестве значения можно использовать `time`

Задание 2

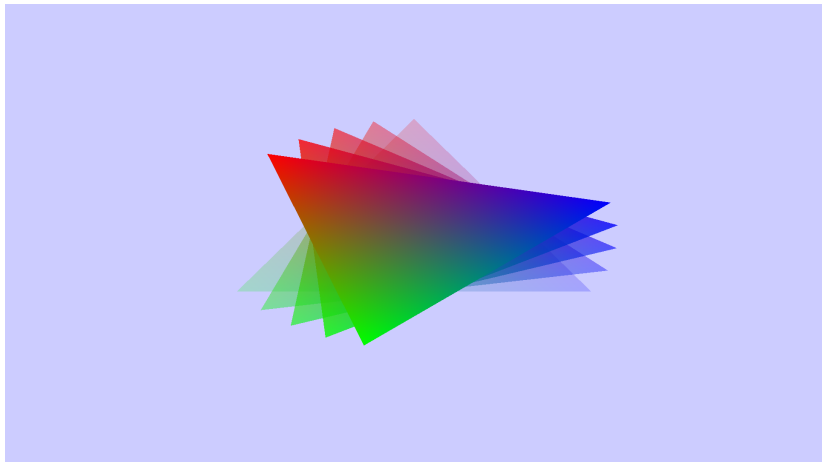


Задание 3

Заменяем ручное применение преобразования на матрицу

- ▶ В коде шейдера:
 - ▶ Заменяем две uniform-переменные на одну:
`uniform mat4 transform`
 - ▶ Заменяем ручное вращение и масштабирование на умножение на матрицу:
`gl_Position = transform * vec4(...);`
- ▶ Обновляем вызов `glGetUniformLocation`
- ▶ В теле основного цикла рендеринга
 - ▶ Создаём матрицу 4×4 - массив из 16 float'ов
`float transform[16] =`
`{`
`?, ?, ?, ?,`
`?, ?, ?, ?,`
`?, ?, ?, ?,`
`?, ?, ?, ?,`
`};`
 - ▶ Заполняем матрицу значениями, чтобы это была матрица применения поворота и масштабирования
 - ▶ `glUniformMatrix4fv, count = 1, transpose = GL_TRUE`

Задание 3

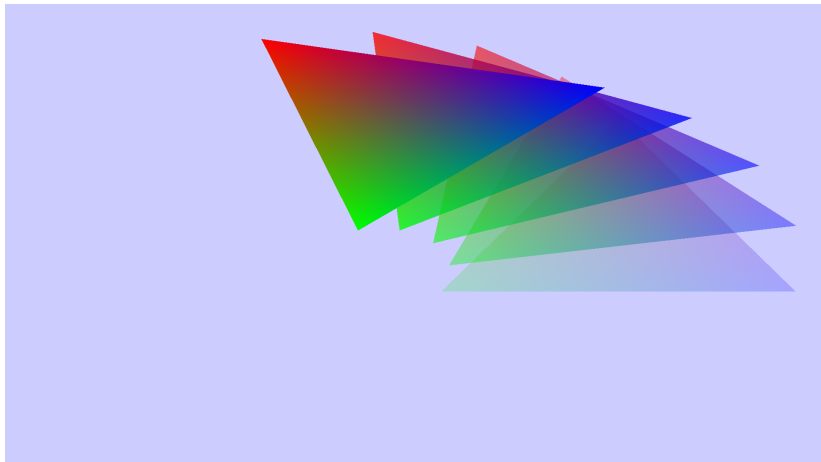


Задание 4

Добавляем в матрицу сдвиг, зависящий от времени

- ▶ В теле основного цикла рендеринга
 - ▶ Заводим переменные под сдвиг:
`float x = ?;`
`float y = ?;`
 - ▶ Обновляем матрицу преобразования:
`float transform[16] = ...;`

Задание 4

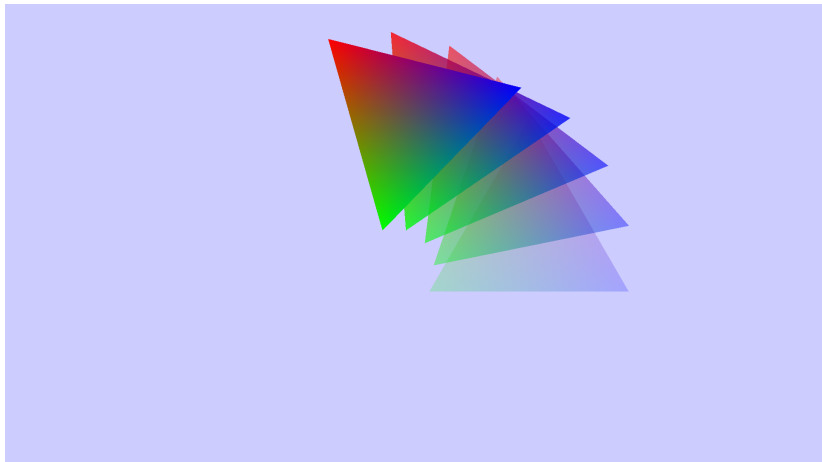


Задание 5

Добавляем учёт aspect ratio экрана

- ▶ В коде шейдера:
 - ▶ Добавляем uniform-переменную `view`, аналогичную переменной `transform`
 - ▶ Применяем обе матрицы:
`gl_Position = view * transform * ...;`
- ▶ После создания программы, до основного цикла:
 - ▶ Добавляем `glGetUniformLocation`
- ▶ В теле основного цикла рендеринга
 - ▶ Вычисляем `aspect_ratio = width / height`
 - ▶ Создаём новую матрицу, которая делит x-координату на `aspect_ratio`
`float view[16] = ...;`
 - ▶ Устанавливаем значение новой uniform-переменной с помощью `glUniformMatrix4fv`

Задание 5



Задание 6

Выключаем VSync

- ▶ В теле основного цикла рендеринга
 - ▶ Выводим в лог значение переменной `dt` (время, потраченное на один кадр в секундах) - скорее всего, будет в районе `0.016`
- ▶ После создания OpenGL-контекста:
 - ▶ `SDL_GL_SetSwapInterval(0);`
 - ▶ Проверяем значение переменной `dt` - должно стать значительно меньше (например, `0.001`)
- ▶ В теле основного цикла рендеринга
 - ▶ Заменяем вычисление `dt` на какую-нибудь константу, например `float dt = 0.016f;`
 - ▶ Должен получиться эффект, похожий на `wagon-wheel effect`
- ▶ N.B.: может не сработать (зависит от системы, драйвера, и т.п.), ничего страшного

Задание 7*

Заменяем треугольник на шестиугольник

- ▶ Нужно поменять количество вершин в вызове `glDrawArrays`
- ▶ Нужно правильно вычислить координаты вершин в вершинном шейдере

Задание 7

