УДК 519.688

АЛГОРИТМЫ ПОСТРОЕНИЯ ФРАКТАЛОВ НА ЯЗЫКЕ ПРОГРАММИРОВАНИЯ ШЕЙДЕРОВ GLSL

канд. техн. наук Д. В. КОЧКИН, М. С. ШАБАНОВ, Д. Р. КОКОША, Д. В. НИКОЛАЕВ
(Вологодский государственный университет, Россия)

Аннотация. *В статье рассматривается алгоритмы построения различных фракталов на языке программирования GLSL, предназначенного для создания шейдеров для спецификации OpenGL. Приведена программная реализация фракталов «Треугольника Серпинского» и «Снежинки Коха».*

Ключевые слова: *GLSL, OpenGL, программирование, алгоритм, фракталы, Треугольник Серпинского, Снежинка Коха.*

Введение. Рассмотрим алгоритмы построения фракталов на примере приложения для построения формульных и хаотичных фракталов. Актуальность темы обусловлена широкими возможностями применения фракталов в различных предметных областях [1, 2, 3] начиная от создания ландшафта в компьютерных играх и заканчивая симуляцией хаоса в физических экспериментах. Разрабатываемое приложение может помочь привлечь внимание студентов и специалистов к теме применения фракталов и повысить уровень компетенций в этой сфере [4].

Результаты и описание работы. Важной особенностью языка GLSL, повлиявшей на процесс разработки программного кода является отсутствие рекурсии. Язык GLSL запрещает её использование в любом виде, что создаёт ряд сложностей при написании алгоритмов.

Рассмотрим алгоритм построения фрактала «Треугольник Серпинского». Строится этот фрактал по такому алгоритму:

1. Строиться равносторонний треугольник (рисунок 1, *а*);
2. Из центра удаляется перевернутый треугольник (рисунок 1, *б*);
3. Далее удаляется три перевернутых треугольника из трех оставшихся треугольников (рисунок 1, *в*);
4. Продолжая этот процесс, на n-ом шаге удаляем 3^{n-1} перевернутых треугольников из центров 3^{n-1} оставшихся треугольников [1].

Реализуется этот алгоритм следующим образом. Сначала идет построение полностью закрашенного треугольника, из которого будут удаляться лишние треугольники. Размер шаблонного образца удаляемых треугольников изменяется в зависимости от номера проводимой итерации, находя левую верхнюю вершину треугольника, находящегося в левом нижнем углу начального треугольника. После

нахождения оставшихся вершин, программа проверяет, принадлежит ли проверяемый пиксель этому треугольнику. Если принадлежит, то ему присваивается фоновый цвет, иначе образец треугольника передвигается вправо, повторяя проверку. Это происходит, пока модель треугольника не выйдет за границы треугольника, после чего проход выполняется снова слева направо, поднимая нашу форму на слой выше, если такая возможность есть.

а *б* *в*

***а* – 0 итераций; *б* – 1 итерация; *в* – 2 итерации**

Рисунок 1. – Построение «Треугольника Серпинского»:

Программный код фрактала «Треугольник Серпинского» представлен на рисунке 2. Для проверки на принадлежность точки треугольнику используется функция check, которая сравнивает позицию точки на плоскости по сравнению с прямыми, образующими треугольник abc. В gl_FragColor передается цвет пикселя в системе RGBA, но значения должны находится в пределе от 0 до 1, а не от 0 до 255.

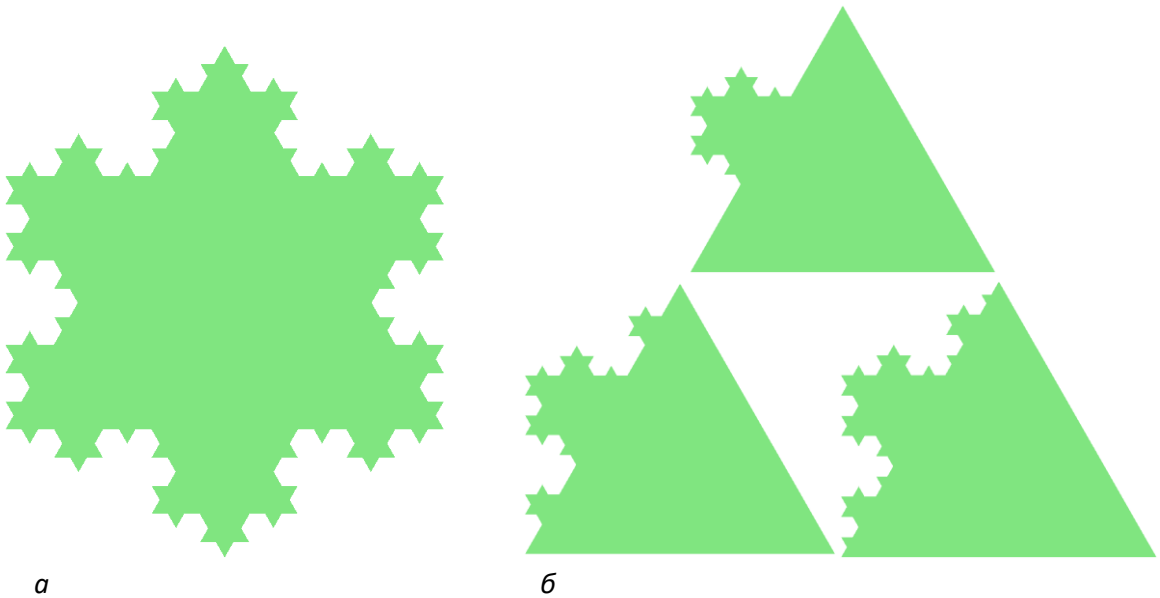
```
"uniform vec2 center;"
"uniform vec2 a_base_triangle;"
"uniform vec2 b_base_triangle;"
"uniform vec2 c_base_triangle;"
"uniform float scale;"
"uniform float k_serp;"
"uniform int iter_serp_triangle;"
"varying highp vec2 coords;"
"uniform vec2 offset;"
"bool check(vec2 a, vec2 b, vec2 c, vec2 z){"
"  float k_ab = -k_serp, k_bc = k_serp, b_ac, b_ab, b_bc;"
"  b_ac = c.y;"
"  b_ab = a.y - k_ab * a.x;"
"  b_bc = c.y - k_bc * c.x;"
"  if(k_ab * z.x + b_ab - z.y <= 0.0 && k_bc * z.x + b_bc - z.y <= 0.0 && b_ac - z.y >= 0.0)"
"      return true;"
"  return false;"
"}"
"void main() {"
"  vec2 z, a, b, c, a_n, b_n, c_n;"
"  z.x = 1.3333 * (coords.x) * scale - center.x - offset.x;"
"  z.y = (coords.y) * scale - center.y + offset.y;"
"  float w = abs(c_base_triangle.x - a_base_triangle.x);"
"  float h = w * cos(radians(30.0f));"
"  float k_ab = k_serp, k_bc = -k_serp, b_ac, b_ab, b_bc;"
"  b_ac = c_base_triangle.y;"
"  b_ab = a_base_triangle.y - k_ab * a_base_triangle.x;"
"  b_bc = c_base_triangle.y - k_bc * c_base_triangle.x;"
```

Рисунок 2. – Код шейдера для фрактала «Треугольник Серпинского» (начало)

```
"  if(k_ab * z.x + b_ab - z.y >= 0.0 && k_bc * z.x + b_bc - z.y >= 0.0 && b_ac - z.y <= 0.0){"
"      gl_FragColor = vec4(0.5, 0.9, 0.5, 1.0);"
"      for(int st = 1; st < iter_serp_triangle; ++st){"
"          a_n.x = a_base_triangle.x + w / pow(2.0, float(st) + 1.0);"
"          a_n.y = a_base_triangle.y + h / pow(2.0, float(st));"
"          b_n.x = a_base_triangle.x + w / pow(2.0, float(st));"
"          b_n.y = a_base_triangle.y;"
"          c_n.x = a_n.x + w / pow(2.0, float(st));"
"          c_n.y = a_n.y;"
"          for(int i = 0; i < int(pow(2.0, float(st)) - 1.0); ++i){"
"              a = a_n;"
"              b = b_n;"
"              c = c_n;"
"              for(int j = i; j < int(pow(2.0, float(st)) - 1.0); ++j){"
"                  if(check(a, b, c, z)){"
"                      gl_FragColor = vec4(1.0, 1.0, 1.0, 1.0);"
"                      break;"
"                  }"
"                  a.x += w / pow(2.0, float(st));"
"                  b.x += w / pow(2.0, float(st));"
"                  c.x += w / pow(2.0, float(st));"
"              }"
"              a_n.x = b_n.x;"
"              a_n.y += h / pow(2.0, float(st));"
"              b_n.x = c_n.x;"
"              b_n.y += a_n.y;"
"              c_n.x += w / pow(2.0, float(st) + 1.0);"
"              c_n.y = a_n.y;"
"          }"
"      }"
"  }"
"  else{"
"      gl_FragColor = vec4(1.0, 1.0, 1.0, 1.0);"
"  }"
"}"
```

Рисунок 2. – Код шейдера для фрактала «Треугольник Серпинского» (окончание)

Рассмотрим далее алгоритм построения фрактала «Снежинка Коха», представленного на рисунке 3. Из-за отсутствия возможности использовать рекурсию приходится осуществлять построение каждой итерации вручную.



а *б*

***а* – построение фрактала на одной стороне; *б* – полное построение фрактала**

Рисунок 3. – Этапы построения «Снежинки Коха» для 4 итераций

Алгоритм построения «Снежинки Коха» следующий:

1. Строится равносторонний треугольник ABC.
2. Сторона AB делится на три равные части – AE, EF, FB.
3. На отрезке EF, как на основании строится равносторонний треугольник EFG.
4. Шаг 3 осуществляется для каждой стороне треугольника ABC, после чего на каждом отрезке новой фигуры повторяются эти действия до достижения заданного количества повторений (рисунок 3, *а*).

Реализация алгоритма основывается на изменении порядка действий для построения фрактала. В начале осуществляется построение всех необходимых треугольников на одной стороне, после этого осуществляется переходя к другой. Это касается каждого строящегося треугольника – строя треугольник 2 итерации, выполняется построение его конечного вида (рисунок 3, *б*).

Функция check работает так же, как и у «Треугольника Серпинского». Функции l_a, l_b, l_c, c_a, c_b, c_c, r_a, r_b и r_c (рисунок 4, *а*) возвращают точки a, b, c для треугольников, основание которых находящихся на левой, нижней (верхней) и правой гранях соответственно.

```
"vec2 l_a(vec2 a, float znak, float st, float w, float h){"
"  a.x = a.x;"
"  a.y = a.y + znak * (h / pow(3.0,st) + h / pow(3.0,st));"
"  return a;"
"}"
"vec2 l_b(vec2 a, float znak, float st, float w, float h){"
"  a.x = a.x + w / (2.0 * pow(3.0, st));"
"  a.y = a.y + znak * (h / pow(3.0, st));"
"  return a;"
"}"
"vec2 l_c(vec2 a, float znak, float st, float w, float h){"
"  a.x = a.x + w / pow(3.0,st);"
"  a.y = a.y + znak * (h / pow(3.0, st) + h / pow(3.0, st));"
"  return a;"
"}"
"vec2 c_a(vec2 a, float znak, float st, float w, float h){"
"  a.x = a.x + w / pow(3.0, st);"
"  a.y = a.y;"
"  return a;"
"}"
"vec2 c_b(vec2 a, float znak, float st, float w, float h){"
"  a.x = a.x + w / (2.0 * pow(3.0, st - 0.1));"
"  a.y = a.y - znak * (h / pow(3.0, st));"
"  return a;"
"}"
"vec2 c_c(vec2 a, float znak, float st, float w, float h){"
"  a.x = a.x + w / pow(3.0, st) + w / pow(3.0, st);"
"  a.y = a.y;"
"  return a;"
"}"
"vec2 r_a(vec2 c, float znak, float st, float w, float h){"
"  c.x = c.x - w / pow(3.0, st);"
"  c.y = c.y + znak * (h / pow(3.0, st) + h / pow(3.0, st));"
"  return c;"
"}"
"vec2 r_b(vec2 c, float znak, float st, float w, float h){"
"  c.x = c.x - w / (2.0 * pow(3.0, st));"
"  c.y = c.y + znak * (h / pow(3.0, st));"
"  return c;"
"}"
"vec2 r_c(vec2 c, float znak, float st, float w, float h){"
"  c.x = c.x;"
"  c.y = c.y + znak * (h / pow(3.0, st) + h / pow(3.0, st));"
"  return c;"
"}"
```

а

```
"  if(iter_koh > 1){"
"      vec2 a_n, b_n, c_n, a_const, c_const;"
"      float w = abs(c.x - a.x);"
"      float h = w * cos(radians(30.0f));"
"      float st = 1.0;"
"      int iter = 2;"
"      float znak = 1.0;"
"      if(b.y < a.y)"
"          znak = -1.0;"
"
"      st = 1.0;"
"      a_n = l_a(a, znak, st, w, h);"
"      b_n = l_b(a, znak, st, w, h);"
"      c_n = l_c(a, znak, st, w, h);"
"
"      if(second(a_n, b_n, c_n,z))"
"          return true;"
"
"      st = 2.0;"
"      iter = 2;"
"      while(iter != iter_koh){"
"          a_const = a;"
"          for(int i = 0; float(i) < pow(3.0, st - 1.0); ++i){"
"              a_n = l_a(a_const, znak, st, w, h);"
"              b_n = l_b(a_const, znak, st, w, h);"
"              c_n = l_c(a_const, znak, st, w, h);"
"              if(iter == 2 && third(a_n, b_n, c_n,z))"
"                  return true;"
"              if(iter == 3 && fourth(a_n, b_n, c_n,z))"
"                  return true;"
"              a_const.x = a_const.x + w / (2.0 * pow(3.0, st - 1.0));"
"              a_const.y = a_const.y + h / pow(3.0, st - 1.0);"
"          }"
"          ++iter;"
"          st += 1.0;"
"      }"
```

б

***а* – определение точек треугольников;**
***б* – построение фрактала на левой грани начального треугольника**

Рисунок 4. – Фрагменты кода шейдера для фрактала «Снежинка Коха»

Заключение. Главной проблемой при разработке алгоритмов на языке GLSL является запрет на использование рекурсии. Для обхода этого ограничения, существует два основных метода. Первый – это преобразование рекурсии в цикл.

Пример применения этого метода показан в алгоритме построения фрактала «Треугольник Серпинского». Результат этого цикла такой же, как был бы при использовании рекурсии. Но есть случаи, когда использование цикла невозможно или же трудно реализуемо, как например построение фрактала «Снежинка Коха». В таких случаях возможно вручную реализовать функции, имитирующие рекурсию. При использовании этого метода существует несколько недостатков. Во-первых, объемность кода из-за постоянного повторения схожих условий с минимальными изменениями. Вторая проблема заключаются в том, что с созданием определенного количеством функций приходит невозможность выполнить необходимую итерацию для определенного начального значения. Как пример, у алгоритма построения «Снежинки Коха» нет возможности построить 7 итерацию фрактала, так как существует лишь 4 функций.

Однако несмотря на описанные сложности применение языка GLSL позволяет существенно ускорить процесс отображения фракталов и повысить интерактивность приложения за счет сокращения времени отклика.

ЛИТЕРАТУРА

1. Пугачев, Д. В. Фракталы. Компьютерное моделирование фракталов / Д. В. Пугачев, А. А. Тарасов // Актуальные научные исследования в современном мире. – 2021. – № 12-3(80). – С. 109–117. – EDN CWPTVD.
2. Суконщиков, А. А. Принципы построения самоорганизующихся информационно-телекоммуникационных систем. / А. А. Суконщиков, А. Н. Швецов, И. А. Андрианов, Д. В. Кочкин // Вестник Череповецкого государственного университета. – 2021. – № 1 (100). – С. 56–67.
3. Суконщиков, А. А. Модели и методы построения нейро-нечетких интеллектуальных агентов в информационно-телекоммуникационных системах: моногр. / А. А. Суконщиков, И. А. Андрианов, С. В. Дианов, Кочкин Д. В., Швецов А. Н. – Курск: Университетская книга, 2021. – 152 с.
4. Смирнов, Е. И. Повышение учебной мотивации школьников в процессе освоения понятий самоподобного и фрактального множеств на основе принципа фундирования / Е. И. Смирнов, В. С. Секованов, Д. П. Миронкин // Ярославский педагогический вестник. – 2015. – № 3. – С. 37–42. – EDN UXMIDD.