

ГЕОМЕТРИЧЕСКОЕ И КОМПЬЮТЕРНОЕ МОДЕЛИРОВАНИЕ ТЕХНИЧЕСКИХ СИСТЕМ, ЦИФРОВАЯ ПОДДЕРЖКА ЖИЗНЕННОГО ЦИКЛА ИЗДЕЛИЙ

УДК 519.688

Е. В. ПОПОВ, проф. кафедры инженерной графики и информационного моделирования, д-р техн. наук

ЭВОЛЮЦИЯ ПАРАДИГМЫ КОНСТРУКТИВНОЙ ТВЕРДОТЕЛЬНОЙ ГЕОМЕТРИИ (CSG) НА ОСНОВЕ МОДЕЛЕЙ *SIGNED DISTANCE* *FUNCTIONS*

ФГБОУ ВО «Нижегородский государственный архитектурно-строительный университет».

Россия, 603952, г. Н. Новгород, ул. Ильинская, д. 65.

Тел.: (910) 144-10-28; эл. почта: popov_evgen@list.ru

Ключевые слова: SDF (signed distance functions), конструктивная геометрия (CSG), компьютерное моделирование, шейдерные процедуры, программирование на javascript.

Конструктивная геометрия (CSG) давно является одной из основ 3D-моделирования, позволяя создавать сложные объекты с помощью булевых операций на простых примитивах. Однако традиционные реализации, основанные на полигональных сетках или граничных представлениях (B-Rep), известны топологическими ошибками, вычислительной нестабильностью и высокой геометрической сложностью. Данная статья посвящена эволюции парадигмы в сторону полей знаковых расстояний (SDF) как объемной альтернативы традиционному 3D-моделированию на основе CSG. Анализируя последние достижения и представляя две практические реализации программного обеспечения – интерактивный SDF Builder и SDF Viewer – продемонстрировано, как SDF технологии предлагают надежное, математически корректное и вычислительно эффективное решение. В статье рассматриваются теоретические основы SDF, реализация булевых операций с помощью функций min/max и использование Ray Marching для рендеринга. Показано, что моделирование на основе SDF не только устраняет недостатки классической модели CSG, но и открывает новые возможности для генерации процедур в реальном времени, плавного смешивания и бесшовной интеграции с графическими конвейерами, ускоренными на GPU.

Введение

Поиск цифрового представления твердых объектов привел к разработке нескольких взаимодополняющих методов. Среди них конструктивная твердотельная геометрия (CSG) зарекомендовала себя как фундаментальная парадигма с момента ее формализации в конце 1970-х годов [1]. Основная идея CSG проста и понятна: сложные формы строятся путем комбинирования геометрически элементарных моделей – примитивов, таких как сферы, кубы и цилиндры, и прочих – с помощью булевых операций, таких как объединение, пересечение и вычитание. Этот подход отражает промышленные производственные процессы и предлагает конструктору высокоуровневый, интуитивно понятный интерфейс.

Классические реализации CSG обычно опираются на граничное представление (B-Rep) или полигональные сетки [2, 3]. В этих системах часть



определяется ее поверхностями, ребрами и вершинами. Булевы операции выполняются с помощью вычислительных алгоритмов, которые должны вычислять точные пересечения между гранеными моделями, убирать избыточную геометрию и затем повторно триангулировать полученную поверхность. Хотя эти алгоритмы математически корректны, на практике они известны своей уязвимостью. Они подвержены множеству числовых нестабильностей: ошибки точности с плавающей запятой могут привести к разрывам или «негерметичным» сеткам; появление вырожденных треугольников и перевернутых нормалей весьма нередки; а операции над сложными высокоразрешающими сетками могут быть вычислительно сложными для приложений в реальном времени. Кроме того, результатом булевой операции сетки становится статическая сетка, которая теряет историю создания и параметрическую гибкость исходной конструкции, затрудняя последующие правки [4].

Недавние достижения как в аппаратном, так и в программном обеспечении побудили пересмотреть объемные представления для преодоления этих ограничений. В частности, функции знаковых расстояний (*SDF*) стали мощной альтернативой для представления геометрии [5]. Как отмечают Плотников и Костюк в своем сравнительном исследовании 2025 года, «функции *SDF* демонстрируют высокоскоростные операции и гибкость в создании плавных переходов между объектами» по сравнению с их многоугольными аналогами [6]. Это привело к появлению практических систем, таких как *QuickCSGModeling*, которые используют дискретные *SDF* для воксельных булевых операций в реальном времени [7].

В настоящей статье утверждается, что слияние *CSG* с технологией *SDF* представляет собой не просто постепенное улучшение, а фундаментальную революцию в подходе к 3D-моделированию. Мы рассмотрим математическую элегантность *SDF*, которая преобразует булевы операции в простые, дифференцируемые математические функции. Далее анализируются два конкретных программных продукта: интерактивный “*SDF Builder*” и “*SDF Viewer*”. Эти программы, реализованные в *WebGL/Three.js*, служат практическими тестовыми площадками, воплощающими обсуждаемые теоретические принципы. С помощью этого анализа делается попытка предоставить всесторонний обзор моделирования на основе *SDF*, его реализации и его значимых последствий для будущего компьютерного геометрического моделирования и компьютерной графики.

1. Применение *SDF* в *CSG*

1.1. Фундаментальные принципы *SDF*

SDF (*signed distance functions*) – это объемное представление, где значение, хранящееся в каждой точке пространства, обозначает расстояние до ближайшей поверхности, а знак указывает, находится ли точка внутри (отрицательно) или снаружи (положительно) твердого объекта [8, 9]. Формально, для заданной поверхности $\partial\Omega$ *SDF* определяется как:

$$f(\mathbf{p}) = s(\mathbf{p}) \cdot \min_{\mathbf{q} \in \partial\Omega} \|\mathbf{p} - \mathbf{q}\|, \quad (1)$$

где $s(\mathbf{p})$ равно -1, если точка $\mathbf{p}$ находится внутри объекта, и +1, если она находится снаружи. Сама поверхность – это нулевое множество, где $f(\mathbf{p}) = 0$.

Ключевое свойство, отличающее *SDF* от общих неявных поверхностей, заключается в том, что они удовлетворяют уравнению эйконала $|\nabla f| = 1$, то есть

поле меняется с постоянной скоростью. Это свойство позволяет эффективно создавать алгоритмы рендеринга и обработки, такие как *Ray March*. Как отмечается в онлайн-сообществах, *SDF* можно рассматривать как «3D-процедурную текстуру, преобразуемую в форму 3D-модели», освобождающую проектировщика от «забот о топологии» и «неудачных булевых моделей» [10].

1.2. Сведение CSG к простой математике

В области многоугольного моделирования булева операция является сложной геометрической задачей. В области *SDF* это сводится к тривиальной математической операции [11, 12]. Для двух *SDF* $f_A(\mathbf{p})$ и $f_B(\mathbf{p})$, представляющих два твердых тела, основные операции CSG реализуются следующим образом:

Union: $f_{A \cup B}(\mathbf{p}) = \min(f_A(\mathbf{p}), f_B(\mathbf{p}))$,

Intersection: $f_{A \cap B}(\mathbf{p}) = \max(f_A(\mathbf{p}), f_B(\mathbf{p}))$,

Subtraction ($A - B$): $f_{A \setminus B}(\mathbf{p}) = \max(f_A(\mathbf{p}), -f_B(\mathbf{p}))$.

Данное представление является предельно простым, и в то же время весьма эффективным (см. рис. 1).

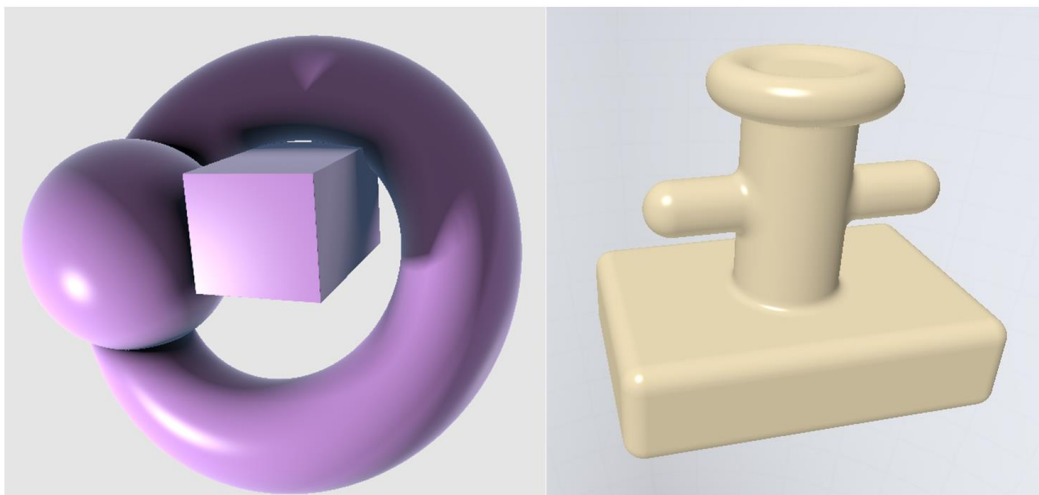


Рис. 1. Типичные примеры применения *SDF* в CSG

Оно не зависит от разрешения, бесконечно масштабируемо по сложности и полностью свободно от топологических ошибок, которые часто возникают в системах, оперирующих полигональными структурами. Кроме того, это позволяет создавать эффекты, которые чрезвычайно сложно достичь с помощью полигональных моделей, например, плавные переходы. Простая модификация операции объединения – гладкий минимум $\text{sm}(a, b, k)$ – создает бесшовное, филлетированное соединение между двумя объектами, параметризуемое коэффициентом смешивания k [9]. Эта математическая простота лежит в основе революции *SDF*.

1.3. Теоретические и вычислительные преимущества

Переход от грани-центричного взгляда к объемно-центричному взгляду дает значительные преимущества. Дерево CSG – бинарное дерево, где листовые узлы являются примитивами, а внутренние – операциями, которые больше не связаны с алгоритмами построения сеток. Вместо этого она становится прямой,



исполняемой функцией. Рендеринг этого дерева *CSG* достигается с помощью *Ray Marching* – варианта трассировки лучей, предназначенной для неявных поверхностей [8]. Луч прокладывается от камеры, и на каждом этапе *SDF* оценивается в текущей точке модели, описываемой математически. Величина *SDF* показывает лучу, пересек ли он поверхность модели или нет, во-первых. И, во-вторых, знак *SDF* позволяет судить о том, находится ли рассматриваемая точка вне, или внутри тела. Этот метод трассировки работает на порядки быстрее, чем стандартные методы поиска корней функций, и идеально подходит для параллельного выполнения на *GPU*.

Как отмечает автор работы [12], «функциональное представление (*F-Rep*) обычно более надежно, чем граничное представление на основе триангуляции», что позволяет «триангулировать модель до *B-Rep* с произвольной точностью» только в качестве финального, необязательного этапа экспорта. Такая отложенная дискретизация сохраняет процесс моделирования гибким и интерактивным, что подтверждается современными инструментами, такими как *MagicaCSG* [13], и программами, обсуждаемыми в следующем разделе.

2. Реализация: *SDF Builder* и *SDF Viewer*

В данном разделе описаны два взаимно дополняющих веб-приложения, построенные на использовании *Three.js* и языка шейдинга *WebGL*. Первая, “*SDF Builder*” (рис. 2), – это интерактивная среда моделирования. Второй, “*SDF Viewer*”, – это специализированный инструмент для загрузки, просмотра и генерации 2D-проекций *SDF*-моделей. Вместе они предназначены для организации полного конвейера для проектирования и презентации на основе *SDF*.

2.1. Архитектура системы и структуры данных

Оба приложения имеют общую основную архитектуру, основанную на динамическом дереве *CSG* [1]. Базовая структура данных – это класс ‘*CSGNode*’, который может быть либо ‘примитивом’ (содержащим индекс конкретного примитива в глобальном массиве), либо ‘операцией’ (содержащей тип операции, например ‘объединение’, ‘вычитание’, ‘пересечение’ и ‘гладкое сопряжение’). Эта структура позволяет строить произвольно сложные деревья.

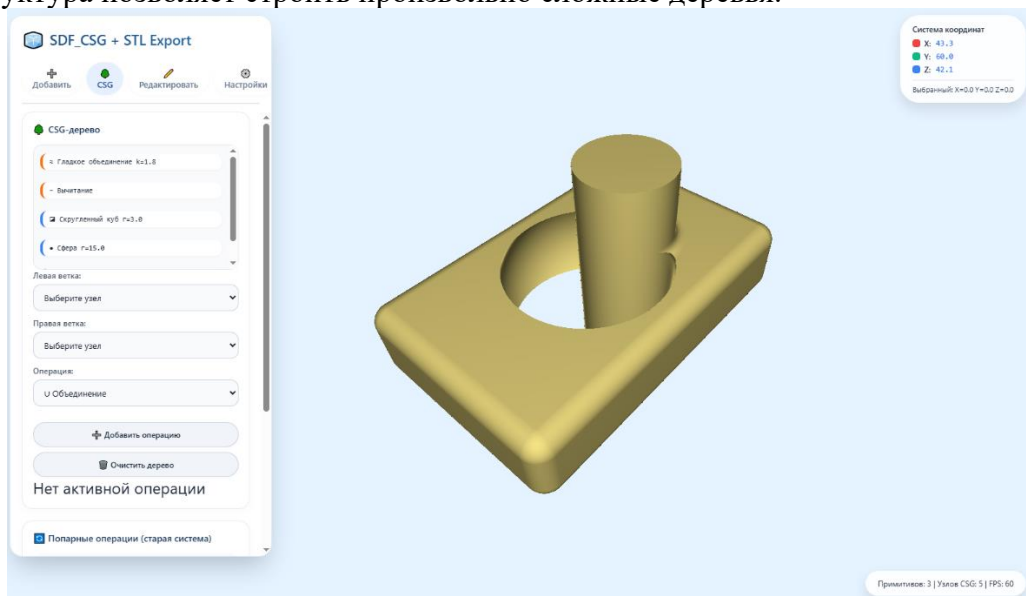


Рис. 2. Общий вид интерфейса *SDF Builder*

Когда пользователь строит модель в *SDF Builder*, в приложении формируется дерево в памяти *JavaScript*. Ключевой шаг – собрать это дерево в шейдер. Рекурсивная функция *'generateSDFCodeFromNode'* проходит по дереву *CSG*. Для каждого примитивного узла она вызывает специфическую для типа функцию (например, *'generatePrimitiveSDFCode'*), которая выводит *GLSL*-код для *SDF* примитива, включая расположение, масштаб и параметры. Для каждого узла операции она объединяет код от левого и правого потомков с помощью соответствующего математического оператора (*'min'*, *'max'* или пользовательской функции *'smin'*). В результате получается одна монолитная функция *GLSL*, представляющая всю модель.

2.2. Ядро шейдеров *Ray Marching* и *Evaluation*

Финальный фрагментный шейдер в обоих приложениях – это место, где происходит главная реализация возможностей *SDF*. Основная функция – *'map(vec3 p)'*, которая благодаря шагу генерации кода содержит полное определение сцены в *SDF*. Цикл рендеринга реализован в функции *'rayMarch'*, классической реализации алгоритма [8] трассировки *Ray Marching*:

```
// Raymarching
float rayMarch(vec3 ro, vec3 rd, float maxDist) {
    float t = 0.0;
    for (int i = 0; i < 100; i++) {
        vec3 p = ro + rd * t;
        float d = map(p);
        if (d < 0.001 || t > maxDist) break;
        t += d;
    }
    return t;
}
```

Как только луч достигает поверхности ($d < \epsilon$), становится известна точка пересечения p . Нормаль в этой точке вычисляется с использованием формулы для градиента *SDF* [6, 8]:

$$\mathbf{n} = \nabla f(\mathbf{p}) = \begin{pmatrix} f(p_x + \epsilon, p_y, p_z) - f(p_x - \epsilon, p_y, p_z) \\ f(p_x, p_y + \epsilon, p_z) - f(p_x, p_y - \epsilon, p_z) \\ f(p_x, p_y, p_z + \epsilon) - f(p_x, p_y, p_z - \epsilon) \end{pmatrix} \quad (2)$$

Эта нормаль затем используется в стандартной модели освещенности Фонга для вычисления конечного цвета пикселя. Кроме того, *SDF Builder* позволяет пользователям менять типы материалов (матовый, глянцевый, металлический) и регулировать яркость в реальном времени, демонстрируя гибкость подхода на основе шейдеров.

2.3. Функциональность и взаимодействие с пользователями

SDF Builder предоставляет комплексный интерфейс для построения деревьев *CSG*. Пользователи могут:

- Добавить широкий спектр примитивов (сфера, коробка, тор, цилиндр, конус, плоскость, капсула, эллипсоид, а также кривые и поверхности Безье).

- Редактировать положения, масштабы и конкретные параметры примитивов (например, радиусы, высоты) с помощью интуитивно понятных ползунков.

- Визуально строить дерево *CSG*, выбирая узлы из выпадающих списков, выбирая операцию и добавляя ее в дерево. Структура дерева визуализируется, и узлы могут быть выбраны для редактирования или удаления.

SDF Viewer позволяет конструктору сосредоточиться на презентации и экспорте. Он загружает сцены *'sdf'* и отображает их с аналогичным качеством рендеринга. Его особенностью является модуль “*2D Orthogonal Projections*”, который использует ту же оценку *SDF* для создания идеальных видов спереди, сзади, сверху, снизу, слева и вправо. Как показывает реализация в классе *'ProjectionGenerator'*, она создает новую ортографическую камеру, выравнивает ее по нужной оси и рендерит *SDF*-сцену на холст за кадром. Это позволяет легко создавать технические чертежи или спрайт-листы непосредственно из *SDF*, что требовало бы сложного сопоставления камер и извлечения карты глубины в процессе полигонального процесса (рис. 3).

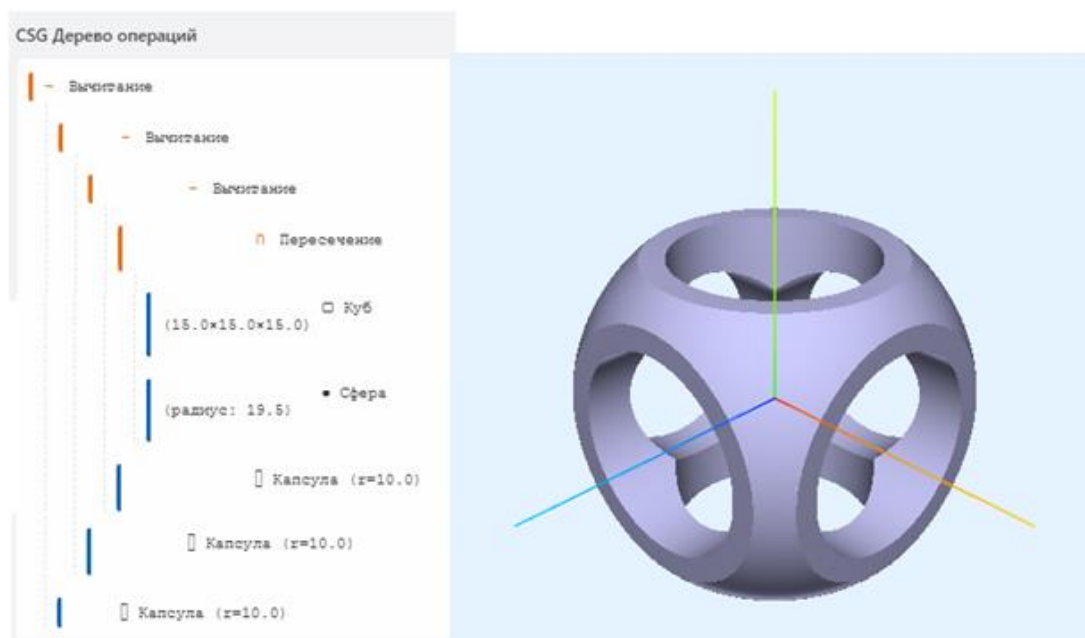


Рис. 3. 3D-модель и *CSG* дерево ее построения

2.4. Формат хранения *CSG* модели на основе *SDF*

При создании 3D-моделей в любой *CAD*-системе конструктор преследует две цели. Во-первых, получение наглядной визуализации модели, с возможностью выполнить операции создания элементов модели, редактирования их, измерения, и, во-вторых, сохранения модели на носителе, формирование на ее основе чертежей, экспорт модели для 3D-принтинга и пр. Во всех этих случаях его не интересует, как правило, математическая основа модели и ее способы представления в памяти компьютера. Это все определяет тот факт, что практическая ценность систем моделирования во многом определяется возможностью сохранять/экспортировать созданные сцены для последующего редактирования, обмена или преобразования в другие форматы. Причем, объемы сохраняемых или экспортируемых файлов по возможности не должны быть

чрезмерно большими. В приложениях “*SDF Builder*” и “*SDF Viewer*” для этой цели был разработан специализированный формат файлов с расширением *.sdf*, основанный на *JSON*-представлении дерева *CSG*. Данный формат обеспечивает компактное, гибкое и расширяемое хранение моделей, полностью сохраняя их параметрическую историю и структуру (рис. 4).

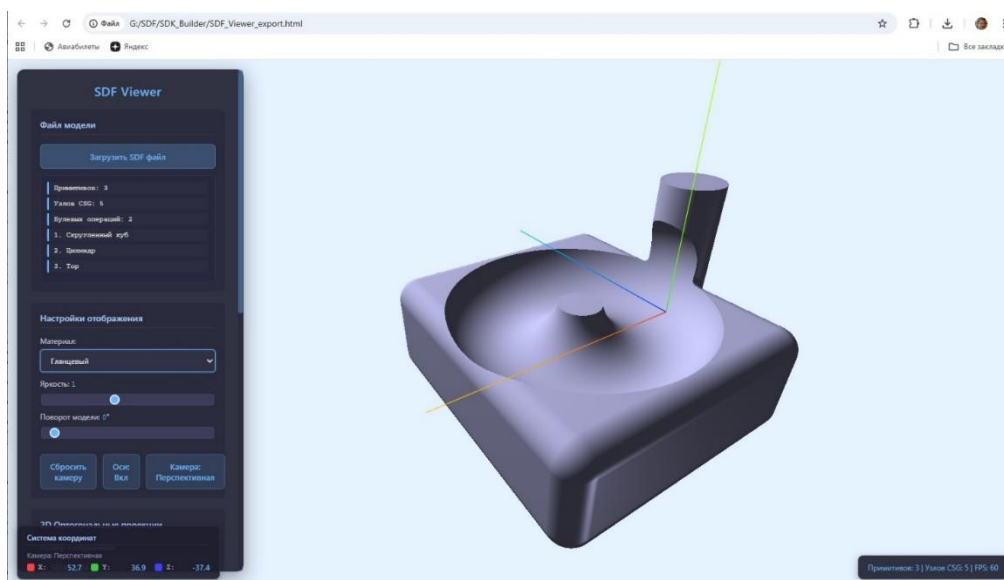


Рис. 4. Общий вид интерфейса *SDF Viewer*

2.5. Структура файла *.sdf*

Файл *.sdf* представляет собой текстовый *JSON*-документ, содержащий три основных раздела: массив примитивов (*primitives*), массив узлов дерева (*csgNodes*) и идентификатор корневого узла (*csgRootId*). Дополнительно присутствует поле *version* для обеспечения обратной совместимости. В дальнейшем планируется расширить количество полей за счет добавления ряда дополнительных свойств модели (материал, масса и пр.) Файл данного формата может быть легко редактируем обычным текстовым редактором.

Примитивы (primitives)

Массив *primitives* содержит описания всех базовых геометрических форм, использованных в модели. Каждый примитив описывается объектом со следующими ключевыми полями:

type – тип примитива (например, “*sphere*”, “*box*”, “*torus*”, “*bezierCubic*”, “*capsule*”, “*roundBox*” и др.);

position – трехмерный вектор с координатами центра примитива (поля *x*, *y*, *z*);

scale – равномерный масштаб примитива;

params – объект с типовыми параметрами формы (радиус сферы, размеры параллелепипеда, координаты контрольных точек для кривых Безье и т. п.);

id – уникальный строковый идентификатор примитива, используемый для ссылок из узлов дерева.

Такое описание позволяет однозначно восстановить геометрию каждого примитива с учетом его положения и параметров. Например, для тора



указываются большой радиус (*radius*) и радиус трубки (*tube*), а для капсулы – координаты конечных точек сегмента (*pointA*, *pointB*) и радиус.

Дерево построения (csgNodes)

Массив *csgNodes* определяет иерархическую структуру модели в виде узлов двух типов: примитивных ("*type*": "*primitive*") и операционных ("*type*": "*operation*"). Каждый узел содержит следующие поля:

id – уникальный идентификатор узла;

type – тип узла ("*primitive*" или "*operation*");

data – дополнительные данные в зависимости от типа:

для примитива: *primitiveIndex* – индекс соответствующего примитива в массиве *primitives*;

для операции: *operation* – тип операции ("*union*", "*intersection*", "*subtraction*", "*smoothUnion*" и др.) и необязательный параметр сглаживания *smoothParam*;

leftId и *rightId* – идентификаторы дочерних узлов (для операции) или *null* (для примитива);

parentId – идентификатор родительского узла, что упрощает навигацию по дереву, хотя топология полностью определяется ссылками *leftId/rightId*.

Такая структура представляет собой бинарное дерево, где листья соответствуют примитивам, а внутренние узлы – булевым операциям. Корень дерева задается полем *csgRootId*. Наличие отдельных массивов для примитивов и узлов позволяет эффективно переиспользовать геометрию: один и тот же примитив может входить в разные ветви дерева (хотя в текущей реализации это не применяется, формат поддерживает такую возможность).

3. Описание алгоритма

Программа *SDF Builder* реализует визуализацию в реальном времени на основе полей знаковых расстояний (*SDF*) с использованием конструктивной сплошной геометрии (*CSG*). Алгоритм состоит из трех основных частей: представление геометрических примитивов и их *CSG*-комбинаций, генерация составного выражения *SDF* и рендеринг методом маршировки лучей с освещением.

3.1. SDF-примитивы

Каждый примитив задается функцией знакового расстояния $f_i(\mathbf{p})$, которая возвращает расстояние от точки $\mathbf{p} \in \mathbb{R}^3$ до поверхности примитива. Поддерживаются следующие примитивы:

Сфера: $f(\mathbf{p}) = \|\mathbf{p} - \mathbf{c}\| - r$

Куб: $f(\mathbf{p}) = \text{sdBox}(\mathbf{p} - \mathbf{c}, \mathbf{b})$, где $\mathbf{b}$ – вектор *SDF*.

Скругленный куб: $f(\mathbf{p}) = \text{sdRoundBox}(\mathbf{p} - \mathbf{c}, \mathbf{b}, r)$

Тор: $f(\mathbf{p}) = \text{sdTorus}(\mathbf{p} - \mathbf{c}, (R, r))$

Цилиндр: $f(\mathbf{p}) = \text{sdCylinder}(\mathbf{p} - \mathbf{c}, r, h)$

Конус: $f(\mathbf{p}) = \text{sdCappedCone}(\mathbf{p} - \mathbf{c}, h, r_1, r_2)$

Плоскость: $f(\mathbf{p}) = \mathbf{n} \cdot \mathbf{p} + d$

Капсула: $f(\mathbf{p}) = \text{sdCapsule}(\mathbf{p}, \mathbf{a}, \mathbf{b}, r)$

Эллипсоид: $f(\mathbf{p}) = \text{sdEllipsoid}(\mathbf{p} - \mathbf{c}, \mathbf{r})$

Кубическая кривая Безье: расстояние до трубчатой кривой, заданной четырьмя контрольными точками.

Поверхность Безье: расстояние до бикубической поверхности Безье с толщиной.

Все примитивы поддерживают перенос (позиция $\mathbf{c}$) и равномерное масштабирование s . Расстояние для масштабированного примитива вычисляется как $s \cdot f((\mathbf{p} - \mathbf{c})/s)$.

3.2. Представление CSG-дерева

Сцена организуется в виде бинарного дерева (CSG-дерева), где листьями являются примитивы, а внутренними узлами – булевы операции. Каждый узел представляет собой экземпляр класса *CSGNode* со следующими атрибутами:

type: либо 'primitive', либо 'operation';

data: для примитивов хранит индекс в массиве примитивов; для операций хранит тип операции и дополнительные параметры (например, коэффициент гладкости k);

left, right: дочерние узлы;

parent: родительский узел (для навигации).

Поддерживаются следующие операции:

Объединение: $\min(f_1, f_2)$

Вычитание: $\max(f_1, -f_2)$ ($A - B$) или *swapped*

Пересечение: $\max(f_1, f_2)$

Гладкое объединение: $\text{sm}in(f_1, f_2, k)$, где *sm*in – полиномиальный гладкий минимум.

Дерево строится интерактивно: пользователь выбирает два узла (примитивы или существующие операционные узлы) и операцию; создается новый узел операции и добавляется в список узлов. Корнем дерева считается любой узел без родителя. Если существует несколько корневых узлов, они неявно объединяются.

3.3. Генерация составного SDF-выражения

Для визуализации сцены программа генерирует единое *GLSL*-выражение, вычисляющее знаковое расстояние в любой точке $\mathbf{p}$. Это делается рекурсивным обходом CSG-дерева:

1. Для узла-примитива генерируется соответствующий *SDF*-код с использованием типа примитива, позиции, масштаба и параметров. Например, для сферы с центром $\mathbf{c}$, радиусом r и масштабом s получаем:

$$\text{sdSphere}((\mathbf{p} - \mathbf{c})/s, r) \cdot s.$$

2. Для узла-операции выражения левого и правого поддеревьев комбинируются с помощью соответствующей *GLSL*-функции:

Объединение: $\min(\text{left}, \text{right})$

Вычитание: $\max(\text{left}, -\text{right})$

Пересечение: $\max(\text{left}, \text{right})$

Гладкое объединение: $\text{sm}in(\text{left}, \text{right}, k)$

Если CSG-дерево отсутствует (т. е. есть только примитивы без операций), используется простой минимум по всем примитивам:

$$f(\mathbf{p}) = \min_i f_i(\mathbf{p}).$$

Сгенерированный код вставляется в функцию *map* фрагментного шейдера. Шейдер также содержит реализации всех *SDF*-примитивов и вспомогательных функций, таких как *sm*in.

3.4. Ray Marching

Рендеринг выполняется методом маршировки лучей (*sphere tracing*) во фрагментном шейдере. Для каждого пикселя вычисляются начало луча $\mathbf{o}$ (позиция



камеры) и направление $\mathbf{d}$ (нормализованный вектор от камеры к позиции фрагмента в мировых координатах). Луч итеративно продвигается:

```
 $t \leftarrow 0$   
for  $i = 1$  to 100 :  
     $\mathbf{p} \leftarrow \mathbf{o} + t\mathbf{d}$   
     $\mathbf{d} \leftarrow \text{map}(\mathbf{p})$   
    if  $d < \epsilon$  or  $t > T_{\max}$  then break  
     $t \leftarrow t + d$ 
```

Если поверхность достигнута ($t < T_{\max}$), точка $\mathbf{p}$ используется для освещения.

3.5. Нормаль и освещение

Нормаль к поверхности оценивается численным дифференцированием *SDF*:

$$\mathbf{n} = \frac{\nabla f(\mathbf{p})}{\|\nabla f(\mathbf{p})\|}, \nabla f(\mathbf{p}) \approx \frac{1}{\epsilon} \begin{pmatrix} f(\mathbf{p} + (\epsilon, 0, 0)) - f(\mathbf{p} - (\epsilon, 0, 0)) \\ f(\mathbf{p} + (0, \epsilon, 0)) - f(\mathbf{p} - (0, \epsilon, 0)) \\ f(\mathbf{p} + (0, 0, \epsilon)) - f(\mathbf{p} - (0, 0, \epsilon)) \end{pmatrix}.$$

Освещение вычисляется с использованием простого направленного источника света. Диффузная и зеркальная компоненты комбинируются в зависимости от выбранного типа материала (матовый, глянцевый, металлик). Итоговый цвет ослабляется с расстоянием для имитации тумана.

3.6. Пользовательский интерфейс и обновление в реальном времени

Приложение предоставляет интерфейс, построенный на *HTML* и *JavaScript*. Все параметры (размеры примитивов, позиции, *CSG*-операции) настраиваются с помощью ползунков и полей ввода. Любое изменение вызывает регенерацию кода шейдера и перекомпиляцию материала, что приводит к мгновенному обновлению изображения. Сцена может быть сохранена в файл (формат *.sdf*) и загружена из него; файл хранит все *CSG*-дерево и определения примитивов в формате *JSON*.

3.7. Производительность

Для поддержания интерактивной частоты кадров цикл *ray march* ограничен 100 итерациями. Шейдер использует высокую точность и избегает динамического ветвления, где это возможно. Сложные сцены с большим количеством примитивов или глубоким *CSG*-деревом могут влиять на производительность, однако использование древовидного *SDF*-выражения обеспечивает логарифмическое количество операций на точку относительно числа узлов.

3.8. Дополнительные примитивы

Программа включает расширенные примитивы, такие как кривые и поверхности Безье. Их функции расстояния аппроксимируются итеративными методами (например, сканирование сегментов кривой или итерации Ньютона) для нахождения ближайшей точки, после чего вычитается радиус трубки или толщина. Эти аппроксимации реализованы непосредственно в шейдере, что позволяет визуализировать искривленные формы в реальном времени.

4. Примеры и преимущества

В таблице ниже приведены фрагменты текста файла, описывающего модель, приведенную на рис. 3. В левой колонке приведен фрагмент, описывающий два



примитива: куб и сферу. В правой колонке приведен фрагмент текста, описывающий некоторые булевы операции.

Таблица

Фрагменты текста файла модели формата SDF

Примитивы	Операции
<pre>{ "primitives": [{ "type": "box", "position": { "x": 0, "y": 0, "z": 0 }, "scale": 1, "params": { "size": { "x": 15, "y": 15, "z": 15 } } }, { "id": "mjpxdz81s7m7r7o035k" }, { "type": "sphere", "position": { "x": 0, "y": 0, "z": 0 }, "scale": 1, "params": { "radius": 19.5 }, "id": "mjpxeuj4m9r034a0jl" }, { </pre>	<pre> "operations": [], "csgNodes": [{ "id": "mke0139msh07a8ey5m", "type": "primitive", "data": { "primitiveIndex": 0 }, "leftId": null, "rightId": null, "parentId": "mke0139mghek4dcyjd" }, { "id": "mke0139m4dlgk4yf3wi", "type": "primitive", "data": { "primitiveIndex": 1 }, "leftId": null, "rightId": null, "parentId": "mke0139mghek4dcyjd" }, { "id": "mke0139mghek4dcyjd", "type": "operation", "data": { "operation": "intersection", "smoothParam": 0.5 }, "leftId": "mke0139msh07a8ey5m", "rightId": "mke0139m4dlgk4yf3wi", "parentId": "mke01kn6cucs40ab9hg" }, </pre>



В данном примере описано сложное дерево: включает пять примитивов (куб, сфера, три капсулы) и использует операции пересечения и вычитания для формирования полого объекта.

Предложенный формат обладает рядом существенных достоинств:

- **Компактность**: модель описывается минимальным набором параметров, без дискретизации поверхностей.
- **Параметричность**: возможность редактирования путем изменения значений в *JSON* без потери качества.
- **Независимость от разрешения**: итоговая геометрия может быть интерпретирована с любой требуемой точностью на этапе экспорта.
- **Текстовый вид**: *JSON*-формат легко просматривать и модифицировать в текстовом редакторе.
- **Расширяемость**: добавление новых типов примитивов или операций не требует изменения общей структуры.

Благодаря этим свойствам формат *.sdf* служит удобным связующим звеном между интерактивным моделированием и финальным рендерингом или производством. Он полностью сохраняет философию *SDF*-подхода, где дерево *CSG* выступает не статическим артефактом, а живым описанием, готовым к исполнению на *GPU*.

На рис. 5 приведены изображения некоторый простых тестовых 3D-моделей, построенных с помощью моделилера *SDF Builder*. Модели носят чисто тестовый характер и не имеют практического применения, однако хорошо демонстрируют эффективность и возможности данной парадигмы компьютерного моделирования.

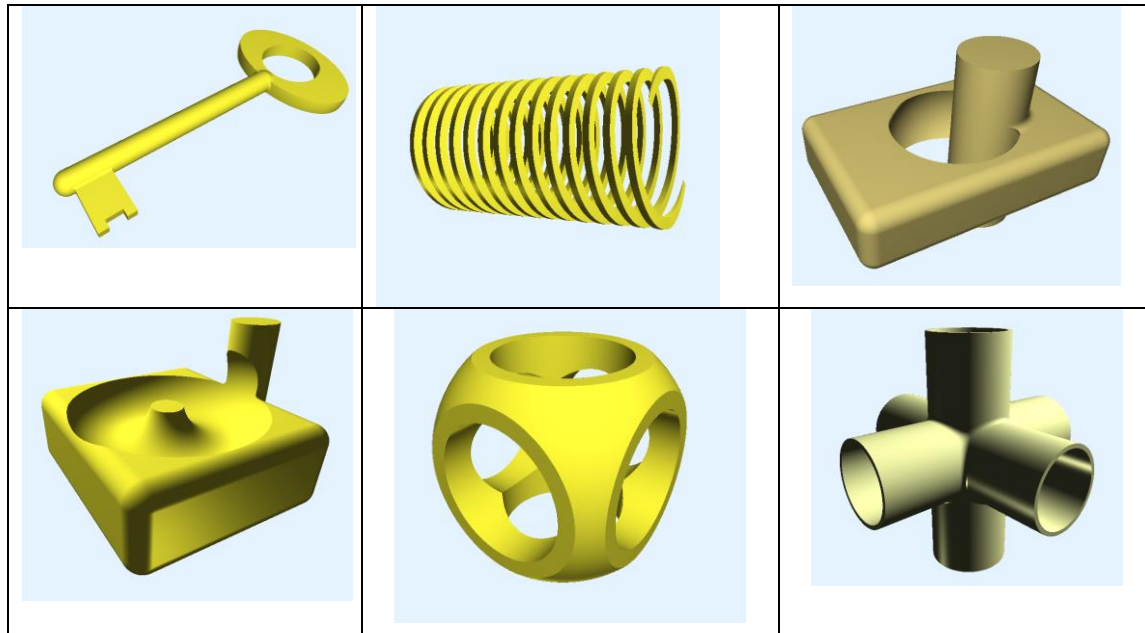


Рис. 5. Некоторые тестовые модели, построенные в среде *SDF Builder*



Заключение

Слияние конструктивной твердотельной геометрии с *SDF* представлением означает значительную эволюцию парадигмы в 3D-моделировании. Замена сложной и уязвимой граничной математики булевых операций над полигональными моделями на надежную, объемную математику *SDF*, решает многие задачи численной устойчивости и топологической корректности. Дерево *CSG* традиционно являясь статической историей построения, превращается в динамичную структуру, исполняемую функцию, которую можно эффективно вычислять и модифицировать средствами *GPU*.

Приложения “*SDF Builder*” и “*SDF Viewer*”, представленные в этой статье, служат практическими демонстрациями этой парадигмы. Они показывают, как относительно небольшое количество кода на *JavaScript* и *GLSL* может создать мощную, гибкую и в реальном времени систему моделирования, способную обрабатывать сложные булевы операции, плавное сглаживание и генерировать стандартные 2D-проекции. Эти инструменты дают возможность конструктору мыслить в терминах чистой геометрии и операций, не обремененных базовой сеткой.

По мере развития аппаратного обеспечения и развития языков шейдеров следует ожидать роста внедрения моделирования на основе *SDF*. Будущие исследования могут изучать интеграцию физических симуляций напрямую в *SDF*, более сложные методы текстурирования и окрашивания, а также разработку гибридных систем, которые бесшовно смешивают объемы *SDF* с традиционными полигональными ресурсами. Революция заключается не в изобретении новой формы, а в открытии нового, более эффективного способа описания самой формы.

БИБЛИОГРАФИЧЕСКИЙ СПИСОК

1. Requicha, A. G. Representations for rigid solids: Theory, methods, and systems / A. G. Requicha // ACM Computing Surveys (CSUR). – 1980. – Vol. 12, № 4. – P. 437–464.
2. Laidlaw, D. H. Constructive solid geometry for polyhedral objects / D. H. Laidlaw, W. B. Trumbore, J. F. Hughes // Proceedings of the 13th annual conference on Computer graphics and interactive techniques. – 1986. – P. 161–170.
3. Mäntylä, M. An introduction to solid modeling / M. Mäntylä. – Rockville, Maryland : Computer Science Press, 1988. – 401 p.
4. Jacobson, A. Robust inside-outside segmentation using generalized winding numbers / A. Jacobson, L. Kavan, O. Sorkine-Hornung // ACM Transactions on Graphics (TOG). – 2013. – Vol. 32, № 4. – P. 1–12.
5. Jones, M. W. 3D distance fields: A survey of techniques and applications / M. W. Jones, J. A. Bærentzen, M. Sramek // IEEE Transactions on Visualization and Computer Graphics. – 2006. – Vol. 12, № 4. – P. 581–599.
6. Plotnikov, I. E. 3D Objects Representation for Real-Time Boolean Operations / I. E. Plotnikov, D. I. Kostyuk // Automatic Documentation and Mathematical Linguistics. – 2025. – Vol. 59, Suppl. 3. – P. S189–S197.
7. QuickCSGModeling: Quick CSG Operations Based on Fusing Signed Distance Fields for VR Modeling / S. Chen, R. Xu, J. Xu [et al.] // ACM Transactions on Multimedia Computing, Communications and Applications. – 2024. – Vol. 20, № 7. – P. 1–18.
8. Hart, J. C. Sphere tracing: A geometric method for the antialiased ray tracing of implicit surfaces / J. C. Hart // The Visual Computer. – 1996. – Vol. 12, № 10. – P. 527–545.
9. Quilez, I. Smooth minimum / I. Quilez // Inigo Quilez's blog. – 2013. – URL: <https://iquilezles.org/articles/smin>.



10. The big SDF thread // Blender Artists Community. – 2024. – URL: <https://blenderartists.org/t/the-big-sdf-thread/1293075>.

11. Constructive solid geometry. SDF vs F-Rep. – URL: https://apm.bplaced.net/w/index.php?title=Constructive_solid_geometry.

12. The big SDF thread (MagicaCSG examples) // Blender Artists Community. – 2024. – URL: <https://blenderartists.org/t/the-big-sdf-thread/1293075/619>.

POPOV Evgeniy Vladimirovich, doctor of technical sciences, professor of the chair of engineering graphics and information modeling

EVOLUTION OF CONSTRUCTIVE SOLID GEOMETRY (CSG) PARADIGM BASED ON SIGNED DISTANCE FUNCTIONS

Nizhny Novgorod State University of Architecture and Civil Engineering.

65, Iljinskaya St., Nizhny Novgorod, 603952, Russia.

Tel.: (910) 144-10-28; e-mail: popov_evgene@list.ru

Key words: SDF (signed distance functions), constructive solid geometry (CSG), computer modeling, shader procedures, JavaScript programming.

Constructive Solid Geometry (CSG) has long been a cornerstone of 3D modeling, enabling the creation of complex objects through Boolean operations on simple primitives. However, traditional implementations based on polygonal meshes or boundary representations (B-Rep) are known to suffer from topological errors, computational instability, and high geometric complexity. This paper discusses the evolution of the paradigm towards signed distance fields (SDFs) as a volumetric alternative to traditional CSG-based 3D modeling. By analyzing recent advancements and presenting two practical software implementations – an interactive SDF Builder and a dedicated SDF Viewer – we demonstrate how SDFs offer a robust, mathematically exact, and computationally efficient solution. The article delves into the theoretical underpinnings of SDFs, the implementation of Boolean operations via min/max functions, and the use of ray marching for rendering. We argue that SDF-based modeling not only resolves the classic pitfalls of mesh-based CSG but also opens new avenues for real-time procedural generation, smooth blending, and seamless integration with GPU-accelerated graphics pipelines.

REFERENCES

1. Requicha A. G. Representations for rigid solids: Theory, methods, and systems. ACM Computing Surveys (CSUR). 1980. Vol. 12, № 4. P. 437–464.
2. Laidlaw D. H., Trumbore W. B., Hughes J. F. Constructive solid geometry for polyhedral objects. Proceedings of the 13th annual conference on Computer graphics and interactive techniques. 1986. P. 161–170.
3. Mäntylä M. An introduction to solid modeling. Rockville, Maryland : Computer Science Press, 1988. 401 p.
4. Jacobson A., Kavan L., Sorkine-Hornung O. Robust inside-outside segmentation using generalized winding numbers. ACM Transactions on Graphics (TOG). 2013. Vol. 32, № 4. P. 1–12.
5. Jones M. W., Bærentzen J. A., Sramek M. 3D distance fields: A survey of techniques and applications. IEEE Transactions on Visualization and Computer Graphics. 2006. Vol. 12, № 4. P. 581–599.



6. Plotnikov I. E., Kostyuk D. I. 3D Objects Representation for Real-Time Boolean Operations. Automatic Documentation and Mathematical Linguistics. 2025. Vol. 59, Suppl. 3. P. 189–197.

7. Chen S., Xu R., Xu J. [et al.] QuickCSGModeling: Quick CSG Operations Based on Fusing Signed Distance Fields for VR Modeling. ACM Transactions on Multimedia Computing, Communications and Applications. 2024. Vol. 20, № 7. P. 1–18.

8. Hart J. C. Sphere tracing: A geometric method for the antialiased ray tracing of implicit surfaces. The Visual Computer. 1996. Vol. 12, № 10. P. 527–545.

9. Quilez, I. Smooth minimum. Inigo Quilez's blog. 2013. URL: <https://iquilezles.org/articles/smin>.

10. The big SDF thread. Blender Artists Community. 2024. URL: <https://blenderartists.org/t/the-big-sdf-thread/1293075>.

11. Constructive solid geometry. SDF vs F-Rep. URL: https://apm.bplaced.net/w/index.php?title=Constructive_solid_geometry.

12. The big SDF thread (MagicaCSG examples). Blender Artists Community. 2024. URL: <https://blenderartists.org/t/the-big-sdf-thread/1293075/619>.

© **Е. В. Попов, 2026**

Получено: 21.06.2026 г.