|  |  |
| --- | --- |
| Gerb-BMSTU_01 | **Министерство науки и высшего образования Российской Федерации**  **Федеральное государственное бюджетное образовательное учреждение**  **высшего образования**  **«Московский государственный технический университет**  **имени Н.Э. Баумана**  **(национальный исследовательский университет)»**  **(МГТУ им. Н.Э. Баумана)** |

|  |  |
| --- | --- |
| ФАКУЛЬТЕТ | «Фундаментальные науки» |
| КАФЕДРА | «Вычислительная математика и математическая физика» |

**РАСЧЕТНО-ПОЯСНИТЕЛЬНАЯ ЗАПИСКА**

***К ВЫПУСКНОЙ КВАЛИФИКАЦИОННОЙ РАБОТЕ***

***НА ТЕМУ:***

|  |
| --- |
| ***«Исследование алгоритмов расчета*** |
| ***координат точек*** |
| ***и производных B-сплайна»*** |
|  |
|  |

|  |  |  |
| --- | --- | --- |
| Студент ФН11-82Б |  | А.Д. Степанов |
|  | (Подпись, дата) | (И.О.Фамилия) |
| Руководитель ВКР |  | А.А. Захаров |
|  | (Подпись, дата) | (И.О.Фамилия) |
| Нормоконтролер |  | С.С. Кудрявцева |
|  | (Подпись, дата) | (И.О.Фамилия) |

*2021 г.*

**РЕФЕРАТ**

Расчетно-пояснительная записка 78 с., 12 рис., 1 табл., 16 источников.

B-СПЛАЙНЫ, СПЛАЙНЫ, NURBS, ПРОФИЛИРОВАНИЕ, АЛГОРИТМ КОКСА – ДЕ БУРА, БАЗИСНЫЕ ФУНКЦИИ

Объектом разработки являются B-сплайновые кривые и поверхности.

Цель работы:

* поиск оптимальных алгоритмов для расчета координат точек и производных B-сплайнов,
* произвести исследование B-сплайновых кривых и поверхностей, их уравнений и свойств;
* выполнить программную реализацию алгоритмов на основе базисных функций и Кокса – де Бура для расчета координат точки кривой и поверхности, алгоритма для вычисления производной в точке кривой и на поверхности;
* произвести анализ производительности работы данных алгоритмов;
* сравнить время выполнения и затраты памяти;
* сделать вывод об оптимальности алгоритмов.

Поставленная цель достигается за счет исследования существующих алгоритмов и сравнения их производительности.

**СОДЕРЖАНИЕ**

[ВВЕДЕНИЕ 5](#_Toc75778172)

[1 Теоретическая часть 7](#_Toc75778173)

[1.1 Сплайновые кривые 7](#_Toc75778174)

[1.2 Происхождение B-сплайна 8](#_Toc75778175)

[1.3 Уравнение B-сплайна 9](#_Toc75778176)

[1.4 Виды B-сплайнов 13](#_Toc75778177)

[1.5 Свойства базисных функций 17](#_Toc75778178)

[1.5.1 Свойство влияния на участке 17](#_Toc75778179)

[1.5.2 Свойство локальности 18](#_Toc75778180)

[1.5.3 Свойство неотрицательности 19](#_Toc75778181)

[1.5.4 Свойство разделенного множества, cвойство производных базисных функций 20](#_Toc75778182)

[1.6 Свойства B-сплайнов 21](#_Toc75778183)

[1.7 Дифференцирование базисных функций B-сплайна 22](#_Toc75778184)

[1.8 Дифференцирование B-сплайн кривой 25](#_Toc75778185)

[1.9 B-сплайн поверхность и ее свойства 28](#_Toc75778186)

[1.10 Дифференцирование B-сплайн поверхности 29](#_Toc75778187)

[1.11 NURBS 31](#_Toc75778188)

[1.12 Алгоритм вычисления базисных функций B-сплайна 33](#_Toc75778189)

[1.13 Алгоритм вычисления B-сплайн кривой 36](#_Toc75778190)

[1.14 Алгоритм вычисления производной базисной функции 40](#_Toc75778191)

[1.15 Алгоритм вычисления производной B-сплайн кривой 43](#_Toc75778192)

[1.16 Алгоритм вычисления B-сплайн поверхности 44](#_Toc75778193)

[1.17 Алгоритм вычисления производных B-сплайн поверхности 47](#_Toc75778194)

[2 Практическая часть 50](#_Toc75778195)

[2.1 Реализация алгоритмов 50](#_Toc75778196)

[2.2 Сравнение характеристик алгоритмов 54](#_Toc75778197)

[ЗАКЛЮЧЕНИЕ 56](#_Toc75778198)

[СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ 57](#_Toc75778199)

[ПРИЛОЖЕНИЕ А Программная реализация алгоритмов 58](#_Toc75778200)

[ПРИЛОЖЕНИЕ Б Графическая часть ВКР 65](#_Toc75778201)

# 

# **ВВЕДЕНИЕ**

В течение последних лет все большее внимание уделялось проблеме геометрического моделирования, то есть определения кривых и поверхностей произвольной формы в интерактивных вычислительных системах.

B-сплайны – это гладкие кривые, которые можно построить для интерполяции или аппроксимации набора контрольных точек. Гладкость кривой B-сплайна определяется существованием и непрерывностью всех производных до -ой включительно на всей области определения [1]. B-сплайны широко используются для проектирования кривых и поверхностей в инженерных и мультимедийных приложениях, являются одними из наиболее часто используемых в геометрическом моделировании. Их распространенность объясняется тем, что они предлагают простые и интуитивно понятные средства интерактивной настройки формы кривой или поверхности.

Стоит отметить и универсальность B-сплайнов. Любой уникальный сплайн, интерполирующий некоторую функцию, можно представить в виде линейной комбинации B-сплайновых функций [2].

Многие работы были посвящены улучшению формы, алгоритмам, позволяющим строить кусочно полиномиальные поверхности на основе B-сплайнов [3], методам подгонки B-сплайнов [4].

Поэтому поиск оптимального алгоритма работы расчета координат точек и производных B-сплайнов является важной и актуальной проблемой.

В данной работе будет изучена литература, посвященная теме B-сплайнов, их производных и алгоритмов их построения. В теоретической части работы будет проведено исследование B-сплайн кривых и поверхностей, их уравнений, свойств и видов. Основное внимание будет сосредоточено на анализе алгоритмов построения B-сплайн кривых и поверхностей, а также их производных.

В практической части работы рассмотренные алгоритмы построения B-сплайн кривых и поверхностей будет выполнена реализация различных алгоритмов, проведен анализ их работы, то есть сравнение времени их выполнения и затрат памяти. На основе этих результатов будут выбраны оптимальные алгоритмы.

# **Теоретическая часть**

Сплайновые кривые

Слово «сплайн» берет свои корни из ремесла постройки лодок, в котором тонким деревянным доскам, называемым шлицами, придавали форму путем сгибания их вокруг колышков, вбитых в землю. Позже так стали называть тонкие гибкие стальные линейки, которые использовали для изображения гладких кривых, проходящих по набору контрольных точек. Совсем недавно термин сплайн-кривая использовался для описания любой кривой, геометрическое место которой контролируется набором фиксированных точек [5]. Гибкий способ построения кривой, форма которой однозначно определяется набором точек, состоит в том, чтобы определить ее геометрическое место как комбинация самих точек. Кривая не обязательно должна проходить через точки или узлы, как их можно иначе называть, скорее, ее форма будет определяться их положением. Сплайн, проходящий через весь набор контрольных точек, называется интерполяционным. Если же некоторые или все контрольные точки не лежат на кривой, то сплайн называется аппроксимационным (рис 1). Как и кривые Безье, о связи с которыми мы поговорим далее, B-сплайны относятся к аппроксимирующим.

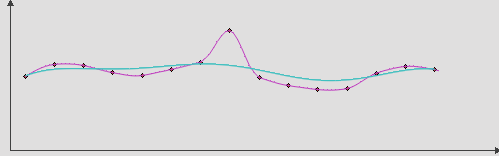


Рисунок 1 – Интерполяция и аппроксимация

Наиболее удобным подходом к описанию кривых и поверхностей является параметрический, поскольку при данном способе однозначно определяются замкнутые кривые, нет сложностей с ее изгибами, поворотами, углами наклона. Рассмотрим упорядоченный список контрольных точек, которые могут быть точками в любом размерном пространстве. Предположим, что они размещены на одномерной параметрической оси через некоторые заданные интервалы. Тогда данные точки могут задавать некоторую параметрическую кривую, вид которой будет определяться из ее уравнения. Далее мы будем пользоваться именно параметрическим способом задания сплайновых кривых.

Происхождение B-сплайна

В 1946 году Исаак Шенберг ввел в математический обиход термин B-сплайна («B» – сокращенно от «basic», то есть базисный). Потребность в введении B-сплайнов возникла в попытке устранить недостатки кривых Безье, которые считались на тот момент одним из лучших способов задания геометрии. Основными недостатками кривых Безье являются невозможность точного представления конических сечений, рост алгебраической степени кривой Безье при аппроксимации кривой, состоящей из большого числа точек. Высокая степень создает большую вычислительную нагрузку на компьютер, которая ощутимо увеличивает время работы алгоритмов. И, наконец, еще одна проблема состоит в трудности локального изменения кривой Безье. Это связано с тем, что форма сплайна зависит от всех точек, задающих данную кривую. Локальные изменения могут приводить к модификации в нужной области, но при этом провоцируют и глобальное изменение формы кривой (рис. 2). Эффект, при котором перемещение контрольной точки влияет только на определенную область кривой называется локальным управлением.

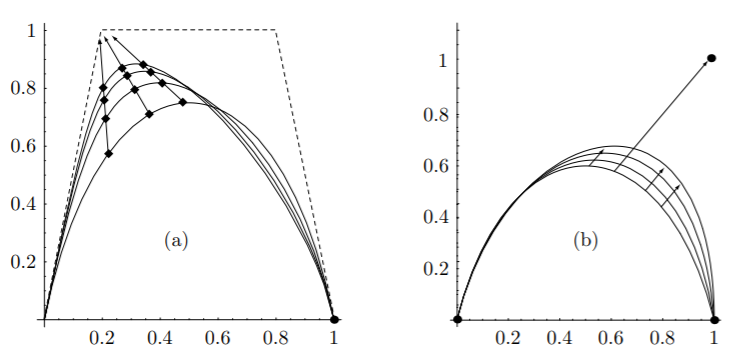


Рисунок 2 – Изменение формы рациональной кривой Безье с помощью (a) – переменной веса и (b) – подвижной точки [6]

B-сплайны стали обобщением кривых Безье и ликвидировали данные сложности, поскольку являются объединением кривых Безье низших порядков. Единственным минусом стало их более сложное математическое описание. Формула расчета B-шлицев выглядит аналогично уравнению кривой Безье, но функциями сопряжения выступают не полиномы Бернштейна, а базисные функции, определяемые рекурсивным соотношением, которое зависит от значения параметра.

Уравнение B-сплайна

Есть несколько способов задания B-сплайна. Например, во многих работах используется прямое описание линейно независимых функций, образующих в совокупности базис, задание с помощью матриц или алгоритм, основанный на условиях гладкости и нормировки [7]. Наиболее полезным в компьютерной реализации является рекуррентный способ задания.

На основе теории Шенберга в 1972 году Кокс и де Бур написали несколько работ, которые установили зависимость между геометрической формой кусочно полиномиального сплайна и его алгебраической формулой, а также предложили новый эффективный алгоритм задания B-сплайнов [8]. Уравнение B-сплайна имеет следующий вид:

|  |  |
| --- | --- |
|  | (1) |

Здесь – базисные функции, которые определяются рекурсивно:

|  |  |
| --- | --- |
|  | (2) |
|  | (3) |

Значения называются узловыми. Они ограничивают отрезки значений параметра, внутри которых базисные функции имеют ненулевые значения. Неопределенность в формуле (3), если она имеется, принимается равной нулю. Это допущение разрешает случай кратности узлов. Из уравнения (3) видно, что для определения базисных функций необходимо задать значений в узлах. Упорядоченную совокупность используемых узлов будем называть узловым вектором. Узловые значения могут быть любыми, но должны представлять собой неубывающую последовательность ( в узловом векторе. От метода задания данных узловых значений будут изменяться базисные функции, а, следовательно, и форма сплайновой кривой [9].

Формула (3) была выведена из определения, данного Карлом де Буром в книге «Практическое руководстве по сплайнам» [10]. В работе де Бура B-сплайны рассматриваются как нормированные -е разделенные разности усеченной степенной функции. Базисные функции определяются правилом:

|  |  |
| --- | --- |
|  | (4) |

Здесь – -я разделенная функции двух переменных и , взятая при фиксированном значении . При этом следует воспринимать как функцию только от .

Теперь рассмотрим формулу Лейбница для -ой разделенной разности произведения для частного случая:

|  |  |
| --- | --- |
|  |  |

Отсюда следует:

|  |  |
| --- | --- |
|  | (5) |

поскольку , , а для .

Тогда можно записать следующее выражение:

|  |  |
| --- | --- |
|  |  |

С учетом данного выражения уравнение (5) можно представить в виде:

|  |  |
| --- | --- |
|  | (6) |

Заменяем в уравнении (6) выражения, соответствующие базисным функциям формулы (4), получаем:

|  |  |
| --- | --- |
|  | (7) |

Заметим, что весовые множители в формуле (7) в сумме всегда дают единицу:

|  |  |
| --- | --- |
|  |  |

При умножении левой и правой части уравнения (7) на мы получаем формулу (3), которая выглядит намного проще и понятнее, чем определение, данное через разделенные разности.

Из формулы (3) также вытекает важное преимущество B-сплайнов перед кривыми Безье, о котором говорилось выше. Так степень кривой Безье определяется количеством задающих ее точек, которые влияют на всю ее форму. В то же время из формулы следует, что степень базисной функции на единицу выше, чем у и . Так как – константа, то – функция первой степени, – второй и так далее. Для порядка базисная функция является линейной комбинацией двух базисных функций -ой степени (рис. 3). Отсюда мы можем сделать вывод, что – это алгебраическая степень B-сплайна.

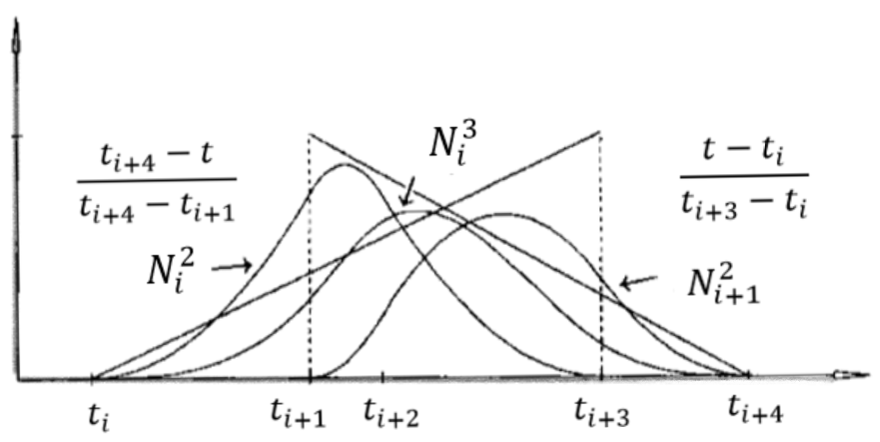


Рисунок 3 – Базисная функция , рассчитанная по рекурсивной формуле Кокса – де Бура [11]

Виды B-сплайнов

Существуют разные методы задания узловых векторов. От вида узлового вектора во многом зависят базисные функции, а, следовательно, и сама кривая. Если расстояние между узловыми значениями постоянно  
(, для )), то B-сплайн, построенный по данному узловому вектору, называется равномерным или однородным. Равномерный вектор узлов может иметь, например, следующий вид:

|  |  |
| --- | --- |
|  |  |

Часто представляется удобным и полезным нормировать диапазон значений от 0 до 1:

|  |  |
| --- | --- |
|  |  |

Еще один распространенный вариант – задание равномерного узлового вектора с нулевым начальным значением и шагом равным 1:

|  |  |
| --- | --- |
|  |  |

У равномерных B-сплайнов базисные функции периодичны. Это означает, что для заданного вектора узлов все базисные функции будут одинаковой формы. Каждая последующая базисная функция является копией предыдущей, смещенной на величину заданного шага (рис. 4) [5]:

|  |  |
| --- | --- |
|  |  |

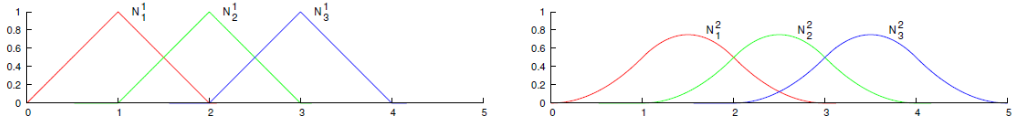


Рисунок 4 – Базисные функции 1-ой и 2-ой степени на равномерном узловом векторе [8]

Работа с равномерными кривыми используется довольно редко, поскольку большинству разработчиков привычнее, чтобы B-сплайн проходил через первую и последнюю контрольную точку, как кривая Безье [9].

В особую категорию выделяют открытые равномерные B-сплайны. Они являются как бы переходным классом от равномерного случая к неравномерному. В узловом векторе у таких B-сплайнов расстояние между узловыми значениями также постоянно, но значения концов повторяются  
 раз. Например, для открытый равномерный вектор узлов может иметь вид:

|  |  |
| --- | --- |
|  |  |

Или, например, нормированный случай для :

|  |  |
| --- | --- |
|  |  |

Построить такой целочисленный узловой вектор можно с помощью правила:

|  |  |
| --- | --- |
|  |  |

Одним из свойств открытых сплайнов является то, что при в векторе узлов есть только кратные значения начала и конца. Тогда базисные функции становятся равными полиномам Бернштейна  
(, ), а сам B-сплайн в данном случае совпадает с кривой Безье.

Если вектор узлов задан произвольными значениями из любых интервалов, удовлетворяя условию , то такой вектор называется неравномерным или неоднородным. Так, например, для неравномерного B-сплайна не только крайние, но и внутренние узловые значения могут быть одинаковыми. Кратные узлы можно использовать, чтобы вытянуть кривую в определенном направлении и создать выступ или даже разрыв в точке соединения.

Если есть потребность генерации неравномерного узлового вектора, то можно, например, задать внутренние значения так, чтобы они были пропорциональны длинам хорд, соединяющих контрольные точки:

|  |
| --- |
|  |

где [12].

Неравномерные B-сплайны обладают высокой гибкостью управления формой кривой. Неравномерность приводит к тому, что базисные функции имеют различную форму (рис. 5). Регулировка значений узлов (а также наличие нескольких одинаковых значений) - это функция, которая помогает точно настроить форму кривой, добиться нужного вида аппроксимации [6].

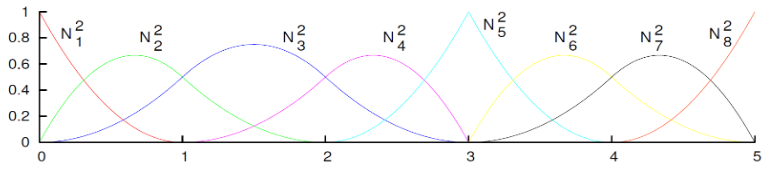


Рисунок 5 – Базисные функции второй степени на неравномерном узловом векторе [13]

Также на форму B-сплайна могут влиять весовые коэффициенты контрольных точек. Если вес точки равен значению , то ее можно задать координатами . Компонент B-сплайна , также как и компоненты , и , является параметрическим B-сплайном и может быть записан следующим образом:

|  |  |
| --- | --- |
|  |  |

Если координаты точки кривой представлены в виде рациональной дроби, то есть

|  |  |
| --- | --- |
|  |  |

то B-сплайн называется рациональным, иначе – нерациональным [14].

Итак, как мы выяснили, выделяют следующие виды B-сплайнов:

* равномерные нерациональные;
* неравномерные нерациональные;
* равномерные рациональные;
* неравномерные рациональные.

Последний тип представляет собой NURBS (non-uniform rational B-splines). Данный вид является наиболее общим способом задания B-сплайнов. Далее данный вид B-сплайнов будет более подробно рассмотрен.

Свойства базисных функций

### **Свойство влияния на участке**

Важную роль в построении B-сплайн кривых и поверхностей играют базисные функции. Поэтому в данном разделе будут подробно рассмотрены некоторые их свойства, которые в дальнейшем помогут сформулировать свойства B-сплайнов в целом.

Любой участок B-сплайна определяется конечным числом контрольных точек, базисные функции в которых будут отличны от нуля. Например, рассмотрим часть кривой, соответствующую полуинтервалу . Из формулы (2) можно видеть, что для нулевой степени от нуля отлична только . Подстановка в правую часть формулы (3) дает ненулевые базисные функции и на рассматриваемом нами промежутке. В первом случае выступает в качестве второго слагаемого, а во втором в качестве первого. Те же действия применяем к найденным ненулевым функциям и , получая функции второго порядка и т.д., доходя, наконец, до степени (рис. 6).

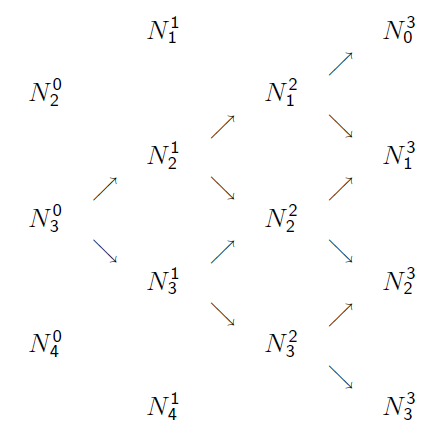
**

Рисунок 6 – Ненулевые базисные функции на полуинтервале [11]

Из сказанного выше очевидно, что ненулевые значения на произвольном промежутке будут иметь только функции . Следственно и влиять на форму заданного сегмента будут только точки (всего точек).

### **Свойство локальности**

Теперь взглянем на базисные функции с другой стороны. Как было замечено ранее, каждая базисная функция , где является комбинацией базисных функций -ого порядка. Однако и они в свою очередь являются комбинацией функций более низкой степени. Таким образом мы можем представить базисную функцию -ого порядка в виде комбинации базисных функций нулевой степени. Последовательно подставляя в левую часть уравнения (3) базисные функции, мы будем находить те функции, из которых они состоят, в конце концов дойдя до базисных функций нулевой степени. В результате мы получим, что каждая базисная функция представляет собой комбинацию значений функций (рис 7). Из (2) следует, что функция не равна нулю на промежутке . Аналогично функция равна единице на полуинтервале и так далее. Тогда можно сделать вывод, что базисная функция имеет ненулевые значения на промежутке . Данное свойство базисных функций называется свойством локальности.

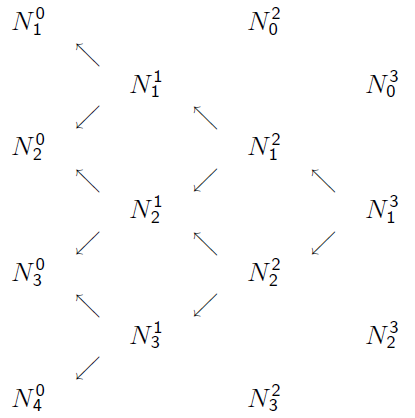


Рисунок 7 – Вычисление через комбинацию базисных функций и [11]

### **Свойство неотрицательности**

Следующим свойством базисных функций является то, что для любых значений , и (свойство неотрицательности). Это может быть доказано по индукции по . Это утверждение очевидно для , поскольку для данного порядка любая базисная функция принимает значение 0 или 1. Предположим, что данное утверждение верно и для , где с произвольными значениями и . По уравнению (3) и по ранее доказанному свойству локальности базисной функции: , если . Но означает, что

|  |  |
| --- | --- |
|  |  |

По нашему предположению, неотрицательно, а потому первая часть уравнения (3) больше либо равна нулю. То же утверждение верно и для второго слагаемого, так как наше предположение было сформулировано для произвольных значений и , а потому мы можем применить к нему все те же утверждения, что и для первого. Таким образом, базисная функция неотрицательна на всей области .

### **Свойство разделенного множества, cвойство производных базисных функций**

Для любого узлового промежутка , для всех . Чтобы доказать данное утверждение, положим

|  |  |
| --- | --- |
|  |  |

Изменяя переменную суммирования во второй сумме с на , и учитывая, что , мы получаем:

|  |
| --- |
|  |

Применяя те же действия, получаем:

|  |
| --- |
|  |

Все производные базисной функции существуют внутри узла, в котором она является полиномом. В данном узле функция будет раз непрерывно дифференцируема, где – кратность этого узла. Из чего следует, что увеличение степени увеличивает непрерывность, а увеличение кратности уменьшает.

Свойства B-сплайнов

Наконец, перечислим все свойства B-сплайнов:

* У B-сплайн кривой степень и непрерывность .
* Если – узловой вектор, то построение B-сплайновой кривой выполняется путем изменения параметра в диапазоне значений узлов . Только в этом интервале сумма базисных функций равна единице ().
* Каждый сегмент кривой (между двумя последовательными значениями узлов) зависит от контрольных точек. Благодаря этому изменения вне данного сегмента не будут на него распространяться.
* Любая контрольная точка участвует определении формы кривой не более чем в сегментах. Это означает, что влияние контрольной точки распространяется только на интервал . Это свойство называется локальной коррекцией (рис. 8).

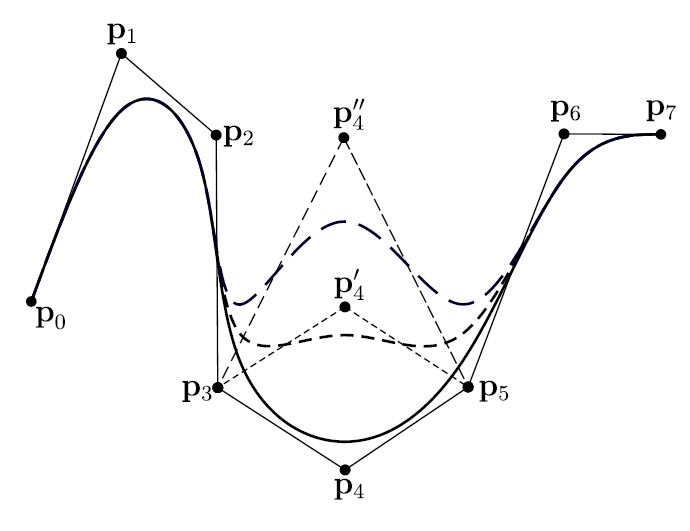


Рисунок 8 – Локальная коррекция B-сплайна [5]

* Уменьшение степени кривой приближает ее к контрольным точкам. Кривые высоких порядков более гладкие.
* При движении вдоль кривой базисные функции сменяют друг друга при прохождении значения параметра очередного узла узлового вектора.
* Локальный контроль может осуществляться изменением числа контрольных точек при построении полиномиальной кривой и без изменения ее степени.
* Всю кривую можно аффинно преобразовать, преобразовав контрольные точки, а затем перерисовав кривую по новым точкам.
* Кривая лежит внутри выпуклой оболочки, определяемой не более чем контрольными точками. Это означает, что кривая проходит близко к контрольным точкам, что позволяет проектировщику легко разместить эти точки, чтобы получить правильную форму кривой [14].

Дифференцирование базисных функций B-сплайна

Производная базисной функции определяется следующим выражением:

|  |  |
| --- | --- |
|  | (8) |

Данное утверждение доказывается методом индукции по . Для этого предположим, что уравнение (8) справедливо для , где . Применяя правило дифференцирования сложной функции к уравнению (3), получаем:

|  |  |
| --- | --- |
|  | (9) |

Подстановка уравнения (8) в уравнение (9) вместо и дает следующее:

|  |  |
| --- | --- |
|  |  |
|  |  |

Преобразуем выражение в скобке:

|  |  |
| --- | --- |
|  |  |

Таким образом, мы получаем:

|  |  |
| --- | --- |
|  |  |

Применяем формулу Кокса-де Бура (3). Получаем:

|  |  |
| --- | --- |
|  |  |

Итак, мы получили то, что требовалось доказать.

Повторно дифференцируя формулу (8), можно получить общую формулу для -ой производной:

|  |  |
| --- | --- |
|  | (10) |

Также -ю производную базисной функции можно посчитать с помощью функций :

|  |  |
| --- | --- |
|  | (11) |

где

|  |  |
| --- | --- |
|  |  |

Дифференцирование B-сплайн кривой

Пусть – B-сплайн кривая степени , определенная на открытом неравномерном узловом векторе , имеющем следующий вид:

|  |  |
| --- | --- |
|  |  |

При фиксированном значении мы можем получить значение производной кривой в данной точке, вычислив для данного базисные функции . Уравнение для производной имеет следующий вид:

|  |  |
| --- | --- |
|  | (12) |

Аналогично для -ой производной уравнение будет иметь следующий вид:

|  |  |
| --- | --- |
|  | (13) |

Однако можно пойти другим путем и не фиксировать параметр , формально продифференцировав B-сплайн кривую степени . Для этот воспользуемся уравнениями (12) и (8):

|  |  |
| --- | --- |
|  |  |

Первое и последнее слагаемые, которые имеют значение , являющееся по принятому для B-сплайнов определению равным 0. Таким образом, получаем:

|  |  |
| --- | --- |
|  |  |

где

|  |  |
| --- | --- |
|  | (14) |

Таким образом, мы получили новый вектор узлов путем исключения первого и последнего узла из узлового вектора :

|  |  |
| --- | --- |
|  |  |

Заметим, что базисная функция , посчитанная на узловом векторе , равна функции , посчитанной на векторе . Тогда получаем уравнение производной кривой B-сплайна в следующем виде:

|  |  |
| --- | --- |
|  | (15) |

Рекурсивное применение формул (14) и (15) дает возможность вычислить производные B-сплайн кривых более высоких порядков:

|  |  |
| --- | --- |
|  | (16) |

где

|  |  |
| --- | --- |
|  |  |

и

|  |  |
| --- | --- |
|  |  |

B-сплайн поверхность и ее свойства

B-сплайн поверхность задается сеткой контрольных точек, для которой выбираются направление и , на которых строятся узловые векторы и соответственно. Векторная функция B-сплайн поверхности представляет собой сумму произведений контрольных точек на декартово произведение базисных функций, с соответствующими точке индексами по параметрическим координатам и :

|  |  |
| --- | --- |
|  | (17) |

Заметим, что значения степеней и по направлениям и соответственно могут не совпадать. В данной работе поверхности, как и кривые, для удобства будут строиться с помощью открытых неравномерных узловых векторов. То есть узловые векторы и имеют следующий вид:

|  |  |
| --- | --- |
|  |  |
|  |  |

B-сплайн поверхности обладают теми же свойствами, что и составляющие их B-сплайны:

* Если , , , , то поверхность будет поверхностью Безье, поскольку базисные функции для таких узловых векторов являются полиномами Бернштейна.
* Поверхность интерполирует четыре угловые контрольные точки: , , , , поэтому из свойства базисных функций о разделенном множестве (для всех ) следует равенство:

|  |  |
| --- | --- |
|  |  |

* Как и кривые, B-сплайн поверхности обладают свойствами аффинной инвариантности, локального улучшения и выпуклой оболочки [11].

Дифференцирование B-сплайн поверхности

В геометрическом моделировании часто необходимо вычислять вектор нормали. Например, это необходимо для того, чтобы геометрический объект обладал правильными отражательными свойствами. Для того, чтобы создать эффект реалистичного освещения изображения поверхности, нужно определять угол между падающим лучом света и нормалью к поверхности в каждой ее точке [9]. Так как вектор нормали можно получить как результат векторного произведения этих производных, то нам необходимо вычислить производные:

|  |  |
| --- | --- |
|  |  |

Дифференцирование по и дает:

|  |  |
| --- | --- |
|  | ( |
|  |  |

Производная B-сплайна считается аналогично формуле (14):

|  |  |
| --- | --- |
|  |  |
|  |  |

Здесь и вычисляются следующим образом:

|  |  |
| --- | --- |
|  |  |
|  |  |

Соответственно узловые векторы будут меняться также как для производных кривых B-сплайнов:

|  |  |
| --- | --- |
|  |  |
|  |  |

Тогда получаем уравнения частных производных B-сплайн поверхности:

|  |  |
| --- | --- |
|  |  |
|  |  |

Общая формула для вычисления производной порядка по параметру и порядка по можно записать следующим образом:

|  |  |
| --- | --- |
|  | (18) |

где

|  |  |
| --- | --- |
|  |  |

NURBS

Наиболее употребляемым видом B-сплайнов является NURBS. Особенностью таких сплайнов является то, что базисные функции, получаемые для неоднородных узлов, применяются к однородным координатам, учитывающим также вес точек [14].

Уравнение NURBS-кривой может быть записано следующим образом:

|  |  |
| --- | --- |
|  | (19) |

Если принять , , а весовой коэффициент для каждого больше нуля (), то базисная функция примет следующий вид:

|  |  |
| --- | --- |
|  |  |

Тогда уравнение можно переписать уже с рациональными функциями B-сплайна:

|  |  |
| --- | --- |
|  | (20) |

Распространяя понятие NURBS-кривых на NURBS-поверхности с помощью декартова произведения, получаем:

|  |  |
| --- | --- |
|  | (21) |

Преимуществом NURBS перед другими видами B-сплайнов является их общность и гибкость. Поэтому в данной работе при построении в практической части мы будем использовать именно NURBS.

Алгоритм вычисления базисных функций B-сплайна

Из свойства влияния на произвольный промежуток базисных функций следует необходимость нахождения индекса в векторе узлов при программировании алгоритма вычисления B-сплайна.

Как отдельный случай следует выделить вычисление при максимально возможном значении параметра . В данной ситуации индекс полагается взять равным . Таким образом, мы получаем, что базисная функция на отрезке .

Вычисление осуществим с помощью алгоритма бинарного поиска. Получим следующую функцию поиска индекса:

**Алгоритм 1:**

int FindSpan (n, k, t, T)

{ /\* Определяет индекс узлового промежутка. \*/

/\* Входные данные:

n – максимально возможный индекс,

k – степень базисной функции,

t – значение параметра искомого промежутка узла,

T – узловой вектор. \*/

/\* Возвращает: индекс узлового промежутка. \*/

if (t == T[n+1])

return n;

/\* Бинарный поиск \*/

low = k;

high = n+1;

mid = (low + high)/2;

while ((t < T[mid]) || (t >= T[mid+1])) {

if (t T[mid])

high = mid;

else

low = mid;

mid = (low + high) / 2;

}

return mid;

}

Теперь нам необходимо понять, как строится алгоритм вычисления базисных функций. Для этого воспользуемся формулой (3) и свойством локальности. Для этого рассмотрим пример. Пусть есть несколько базисных функций второго порядка:

|  |  |
| --- | --- |
|  | (22) |
|  | (23) |
|  | (24) |

По свойству локальности .

Также заметим, что выражение появляется в уравнении (22) и (23). То же относится к выражению .

Введем следующее представление:

|  |  |
| --- | --- |
|  |  |
|  |  |

Тогда уравнения (22) - (24) можно записать так:

|  |  |
| --- | --- |
|  |  |
|  |  |
|  |  |

На основании данного наблюдения можно записать алгоритм, вычисляющий ненулевые базисные функции [11]:

**Алгоритм 2:**

BasisFuns (i, t, k, T, N)

{ /\* Считает значения ненулевых базисных функций. \*/

/\* Входные данные:

i – индекс узлового промежутка,

t – значение параметра,

k – степень базисной функции,

T – узловой вектор. \*/

/\* Возвращает: массив ненулевых базисных функций N. \*/

N[0] = 1.0;

for (j = 1; j <= k; j++)

{

left[j] = t – T[i+1-j];

right[j] = T[i+j]-t;

saved = 0.0;

for (r = 0; r < j; r++)

{

temp = N[r] / (right[r+1] + left[j-r]);

N[r] = saved + right[r+1]⋅temp;

saved = left[j-r]⋅temp;

}

N[j] = saved;

}

}

Данный алгоритм хорош тем, что он не только эффективен, но и вычисляет только ненулевые базисные функции.

Алгоритм вычисления B-сплайн кривой

Перейдем к построению B-сплайн кривой. Приведем здесь алгоритм, который вычисляет ее точку при фиксированном значении параметра . Можно сказать, что данный алгоритм основан на формуле (1). Для построения необходимо сделать три действия:

* найти узловой промежуток, в котором лежит параметр (Алг. 1);
* подсчитать ненулевые базисные функции (Алг. 2);
* умножить значения базисных функций на соответствующие контрольные точки.

Тогда запишем алгоритм [11]:

**Алгоритм 3:**

CurvePoint (n, k, T, P, t, r)

{ /\* Считает точку кривой. \*/

/\* Входные данные:

n – максимально возможный индекс,

k – степень базисной функции,

T – узловой вектор,

P – массив контрольных точек,

t – значение параметра. \*/

/\* Возвращает: значение точки кривой r. \*/

span = FindSpan (n, k, t, T);

BasisFuns (span, t, k, T, N);

r = {0.0, 0.0, 0.0};

for (i=0; i < k; i++)

r += N[i]⋅P[span-k+i];

}

Также для вычисления точки B-сплайн кривой можно воспользоваться алгоритмом Кокса – де Бура. Пусть мы ищем значение точки кривой для параметра , лежащего в узловом диапазоне . Тогда, как и ранее в алгоритме для вычисления ненулевых базисных функций, по свойству влияния на отрезке ненулевыми будут только функции . Тогда мы можем записать формулу для данного промежутка так:

|  |  |
| --- | --- |
|  | (25) |

Подставим формулу (3) в уравнение (25). Получаем:

|  |  |
| --- | --- |
|  | (26) |

Здесь точка определяется выражением:

|  |  |
| --- | --- |
|  |  |

Точку можно воспринимать как внутреннюю точку разбиения сегмента линии .

Применяя аналогичные рассуждения к уравнению (26), получаем:

|  |  |
| --- | --- |
|  |  |
|  |  |

Повторение данной процедуры произвольное число раз , дает формулу:

|  |  |
| --- | --- |
|  |  |
|  |  |

Подставляя в получившуюся формулу , получаем:

|  |  |
| --- | --- |
|  |  |

Таким образом, координаты точки, соответствующей значению параметра в промежутке , – это координаты точки , которые получаются рекурсивным разбиением сегмента линий между соответствующими точками, начиная с контрольных и [9].

Алгоритм Кокса – де Бура может быть реализован следующим образом:

**Алгоритм 4:**

CoxDeBoor (n, k, T, P, t, c)

{ /\* Считает точку кривой. \*/

/\* Входные данные:

n – максимально возможный индекс,

k – степень базисной функции,

T – узловой вектор,

P – массив контрольных точек,

t – значение параметра. \*/

/\* Возвращает: значение точки кривой c. \*/

double left, right;

span = FindSpan (n, k, t, T)

for (j = 0; j < k; j++)

c[j] = P[span-k+1+j];

for (r = 1; r < k; r++)

{

for (j = k – 1; j >= r; j--)

{

i = span-k+1+j;

left = t – T[i];

right = T[i+k-r] – t;

c[j] = (c[j]⋅left + c[j-1]⋅right) / (left + right);

}

}

return c[k-1];

}

Алгоритм вычисления производной базисной функции

Для вычисления производной B-сплайн кривой в фиксированной точке сначала вычислим ненулевые производные базисных функций. Воспользуемся формулой (11) и свойством локальности, чтобы переписать общую формулу, исключив из нее базисные функции равные нулю. Тогда с учетом данного свойства формула примет следующий вид:

|  |  |
| --- | --- |
|  | (27) |

поскольку все функции равны нулю на данном интервале. Аналогично можно воспользоваться данным свойством для других интервалов.

На основе данного замечания сформулируем алгоритм, вычисляющий массив ненулевых базисных функций и их производных ders, где ders[p][j] – -я производная функции , где и [11]:

**Алгоритм 5:**

DersBasisFuns (i, t, k, T, ders)

{ /\* Подсчитывает ненулевые базисные функции и их первую

производную. Первая часть алгоритма является модификацией

алгоритма подсчета базисных функций. Она хранит функции и

разности узлов. \*/

/\* Входные данные:

i – индекс узлового промежутка,

t – значение параметра,

k – степень базисной функции,

T – узловой вектор. \*/

/\* Возвращает: ders \*/

ndu[0][0] = 1.0;

for (j = 1; j <= k; j++)

{

left[j] = t – T[i+1-j];

right[j] = T[i+j] – t;

saved = 0.0;

for (r = 0; r < j; r++)

{

ndu[j][r] = right[r+1] + left[j-r];

temp = ndu[r][j-1] / ndu[j][r];

ndu[r][j] = saved + right[r+1]⋅temp;

saved = left[j-r]⋅temp;

}

ndu[j][j] = saved;

}

/\* Загрузка базисных функций \*/

for (j = 0; j <= k; j++)

ders[0][j] = ndu[j][k];

/\* Часть, считающая производную \*/

for (r = 0; r <= k; r++)

{

s1 = 0;

s2 = 1;

a[0][0] = 1.0;

d = 0.0;

r1 = r – 1;

k1 = k – 1;

if (r >= 1)

{

a[s2][0] = a[s1][0] / ndu[k1+1][r1];

d = a[s2][0]⋅ndu[r1][k1];

}

if (r1 >= -1)

j1 = 1;

else

j1 = -r1;

if (r – 1 <= k1)

j2 = 0;

else

j2 = k – r;

for (j = j1; j <= j2; j++)

{

a[s2][j] = (a[s1][j] – a[s1][j-1]) / ndu[k1+1][r1+j];

d += a[s2][j]⋅ndu[r1+j][k1];

}

if (r <= k1)

{

a[s2][1] = -a[s1][[0] / ndu[k1+1][r];

d += a[s2][1]⋅ndu[r][k1];

}

ders[1][r] = d;

temp = s1;

s1 = s2;

s2 = temp;

}

r = k;

for (j = 0; j <= k; j++)

ders[1][j] \*= r;

}

Алгоритм хорош тем, что избегает деления на ноль и используют значения только из массива ndu[][].

Алгоритм вычисления производной B-сплайн кривой

Рассмотрим алгоритм, который вычисляет производную точки на B-сплайн кривой при фиксированном значении параметра . Данный алгоритм основан на представлении производной B-сплайна с помощью формулы (12).

Для вычисления узлового промежутка все также будет использован Алг. 1, а для производных базисных функций – Алг. 5. Получаем следующий алгоритм:

**Алгоритм 6:**

CurveFirstDeriv (n, k, T, P, t, r1)

{ /\* Считает производную кривой. \*/

/\* Входные данные:

n – максимально возможный индекс,

k – степень базисной функции,

T – узловой вектор,

P – массив контрольных точек,

t – значение параметра. \*/

/\* Возвращает: r1. \*/

span = FindSpan (n, k, t, T);

DersBasisFuns (span, t, k, T, nders);

r1 = {0.0, 0.0, 0.0};

for (j = 0; j < k; j++)

r1 += nder[1][j]⋅P[span-k+j];

}

Алгоритм вычисления B-сплайн поверхности

Приведем алгоритм на основе формулы (17), который вычисляет точку B-сплайн поверхности при фиксированной паре значений параметров . Для этого необходимо выполнить следующие пять шагов:

* найти узловой промежуток , в котором лежит параметр (Алг. 1);
* подсчитать ненулевые базисные функции (Алг. 2);
* найти узловой промежуток ,, в котором лежит параметр s (Алг. 1);
* подсчитать ненулевые базисные функции (Алг. 2);
* умножить значения базисных функций на соответствующие контрольные точки.

Получим следующий алгоритм:

**Алгоритм 7:**

SurfacePoint (n, k, T, m, l, S, P, t, s, R)

{ /\* Считает точку поверхности \*/

/\* Входные данные:

n – максимально возможный индекс параметра t,

k – степень базисных функций по параметру t,

T – узловой вектор по параметру t,

m – максимально возможный индекс параметра s,

l – степень базисных функций по параметру s,

S – узловой вектор по параметру s,

t, s – заданные значения параметров. \*/

/\* Возвращает: точку поверхности R. \*/

spanT = FindSpan (n, k, t, T);

BasisFuns (spanT, t, k, T, Nt);

spanS = FindSpan (m, l, s, S);

BasisFuns (spanS, s, l, S, Ns);

indT = spanT – k;

R = {0.0, 0.0, 0.0};

for (j = 0; j <= l; j++)

{

temp = 0.0;

indS = spanS – l + 1;

for (i = 0; i <= k; i++)

temp += Nt[i]⋅P[spanT+i][spanS];

R += Ns[j]⋅temp;

}

}

Распространим алгоритм Кокса – де Бура на поверхность. Для этого выполним Алг. 4 по направлению , для первых контрольных точек по направлению . К полученной точке снова применяем Алг. 4 теперь в другом направлении. Тогда получаем следующий алгоритм:

**Алгоритм 8:**

SurfacePoint (n, k, T, m, l, S, P, t, s, C)

{ /\* Считает точку поверхности \*/

/\* Входные данные:

n – максимально возможный индекс параметра t,

k – степень базисных функций по параметру t,

T – узловой вектор по параметру t,

m – максимально возможный индекс параметра s,

l – степень базисных функций по параметру s,

S – узловой вектор по параметру s,

t, s – заданные значения параметров. \*/

/\* Возвращает: точку поверхности C[k-1][l-1]. \*/

double left, right;

spanT = FindSpan (n, k, t, T);

spanS = FindSpan (m, l, s, S);

for (i = 0; i < k; i++)

for (j = 0; j < l; j++)

C[i][j] = P[spanT-k+1+i][spanS-l+1+j];

for (j = 0; j < l; j++)

{

for (r = 1; r < k; r++)

{

for (i = k – 1; i >= r; i--)

{

u = spanT – k + 1 + i;

left = t – T[u];

right = T[u+k-r] – t;

C[i][j] = (C[i][j]⋅left + C[i-1][j]⋅right) / (left + right);

}

}

}

for (r = 1; r < k; r++)

{

for (i = k – 1; i >= r; i--)

{

u = spanS – l + 1 + j;

left = s – S[u];

right = S[u+l-r] – s;

C[k-1][j] = (C[k-1][j]⋅left + C[k-1][j-1]⋅right) / (left + right);

}

}

return C[k-1][l-1];

}

Как и алгоритм Кокса – де Бура для кривой, данный алгоритм рекурсивно находит приближенное значение точки, последовательно разбивая участок поверхности.

Алгоритм вычисления производных B-сплайн поверхности

Производные B-сплайн поверхности можно вычислить с помощью следующих формул:

|  |  |
| --- | --- |
|  | ( |
|  |  |

Выполняя дифференцирование соответствующих базисных функций в скобках, получим следующие выражения:

|  |  |
| --- | --- |
|  | ( |
|  |  |

Из уравнений видно, что под знаком суммирования присутствуют множители, алгоритмы которых уже были написаны (алгоритм расчета базисных функций и алгоритм расчета производной базисных функций), поэтому можно записать следующий алгоритм:

**Алгоритм 9:**

SurfaceDeriv (n, k, T, m, l, S, P, t, s, <Rt, Rs>)

{ /\* Считает первую производную по t и s. \*/

/\* Входные данные:

n – максимально возможный индекс параметра t,

k – степень базисных функций по параметру t,

T – узловой вектор по параметру t,

m – максимально возможный индекс параметра s,

l – степень базисных функций по параметру s,

S – узловой вектор по параметру s,

t, s – заданные значения параметров. \*/

/\* Возвращает: пару <Rt, Rs>. \*/

spanT = FindSpan (n, k, t, T);

BasisFuns (spanT, t, k, T, Nt);

DersBasisFuns (spanT, t, k, T, dersT);

spanS = FindSpan (m, l, s, S);

BasisFuns (spanS, s, l, S, Ns);

DersBasisFuns (spanS, s, l, S, dersS);

Rt = {0.0, 0.0, 0.0};

for (i = 0; i <= k; i++)

for (j = 0; j <= l; j++)

Rt += dersT[i]⋅Ns[j]⋅P[spanT-k+i][spanS-l+j];

Rs = {0.0, 0.0, 0.0};

for (i = 0; i <= k; i++)

for (j = 0; j <= l; j++)

Rs += Nt[i]⋅dersS[j]⋅P[spanT-k+i][spanS-l+j];

return < Rt, Rs>

# **Практическая часть**

Реализация алгоритмов

В данном разделе мы осуществим программную реализацию алгоритмов, изложенных в теоретической части. Программный код был написан на языке программирования C++. Для визуализации использовалась графическая библиотека OpenGL, которая предоставляет все необходимые инструменты для работы с 3D-графикой [1].

Для удобства и наглядности алгоритмы теоретической части были записаны для неоднородных нерациональных B-сплайнов. Но, как говорилось ранее, наибольшей вариативностью отличаются NURBS, поэтому все приведенные алгоритмы будут адаптированы, чтобы строить этот тип кривых и поверхностей с их производными.

Для кривых и поверхностей это означает незначительное изменение в алгоритме расчета точек. Вместо представления вектора контрольных точек в виде vector(Vec3D) m\_ControlPoints, хранящего значения {x, y, z}, как это представляется в алгоритме для нерациональных B-сплайнов мы переходим к представлению в виде вектора vector(Vec4D) m\_ControlPoints, который хранит значения {x, y, z, w}, таким образом добавляя информацию о весе точки. Теперь вместо контрольной точки с координатами {}, мы оперируем контрольной точкой, умноженной на весовой коэффициент, то есть с координатами { }. Действия, совершенные с контрольной точкой, применяем и к весовому коэффициенту. Это означает, что при выполнении алгоритма подсчета суммы , тот же алгоритм должен быть применен к сумме . Затем значение первой суммы делится на вторую, что соответствует формуле NURBS-кривой.

Аналогичные изменения затрагивают поверхности. Контрольная точка умножается на весовой коэффициент , рассчитываются суммы и , а затем первая делится на вторую.

Для вычисления производной NURBS-кривой необходимо продифференцировать уравнение (19):

|  |  |
| --- | --- |
|  | (28) |

Из уравнения (28) видно, что для подсчета производной NURBS-кривой имеются все необходимые алгоритмы, которые мы используем в алгоритме 6 (смотреть ПРИЛОЖЕНИЕ А). Для того чтобы убедиться в правильности работы алгоритма визуализируем производную в данной точке в виде касательной (рис. 9). Для поверхностей действия будут аналогичными.

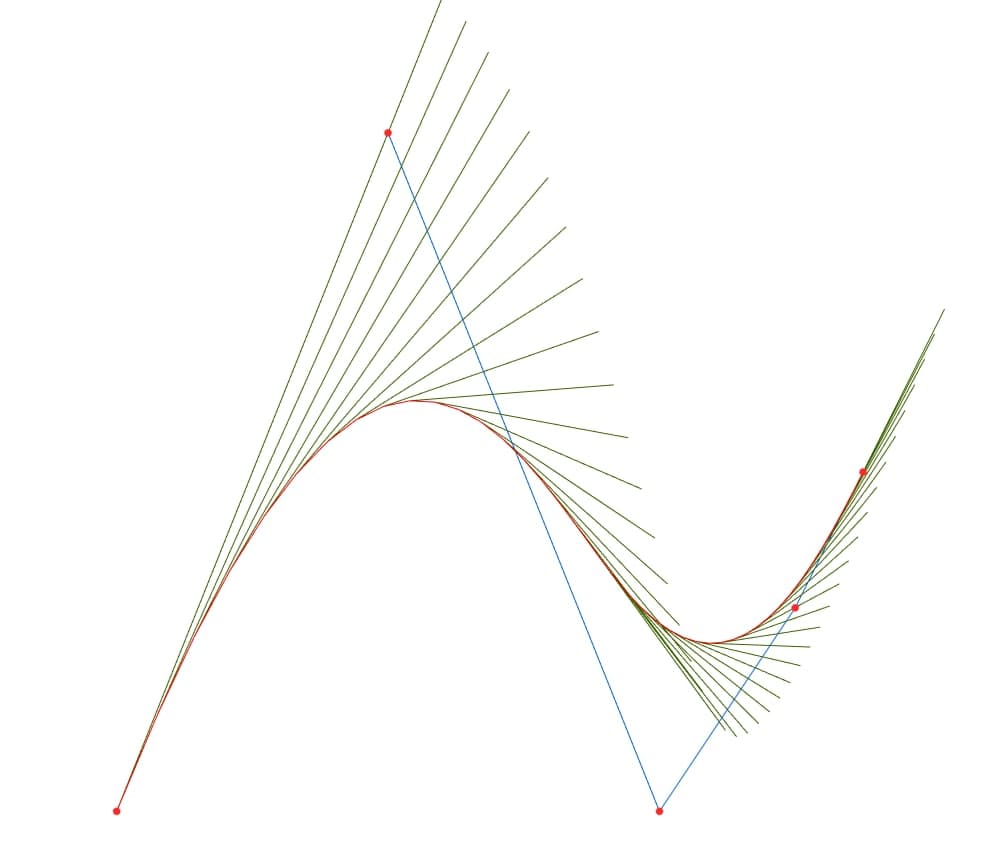


Рисунок 9 – NURBS-кривая и ее первые производные в точках, представленные на рисунке касательными векторами

С помощью адаптированных алгоритмов 7 и 8 (см. ПРИЛОЖЕНИЕ А) построим по заданным контрольным точкам поверхности (рис. 10 и 11).

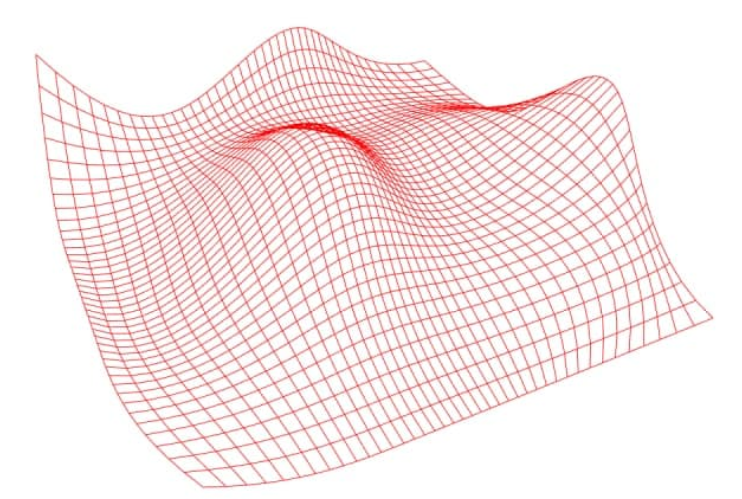


Рисунок 10 – NURBS-поверхность, построенная с помощью алгоритма 7

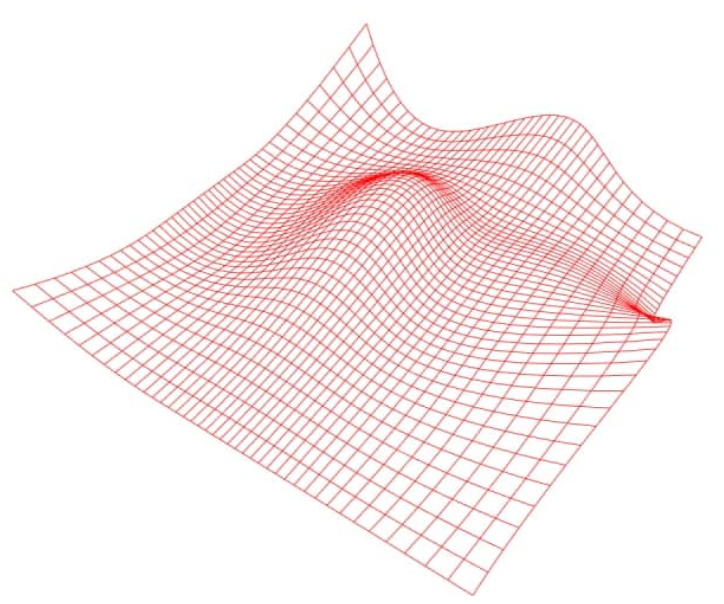


Рисунок 11 - NURBS-поверхность, построенная с помощью алгоритма 8

Так как частные производные и от поверхности по направлению и соответственно с геометрической точки зрения являются касательными по данным направлениям [15]. Тогда по вычисленным с помощью алгоритма 9 первым производным и можно посчитать нормали к поверхности. Нормаль в точке является результатом векторного произведения нормированных касательных векторов:

|  |  |
| --- | --- |
|  |  |

Изображение нормалей наглядно покажет нам правильность работы алгоритма расчета производных и NURBS-поверхности (рис. 11).

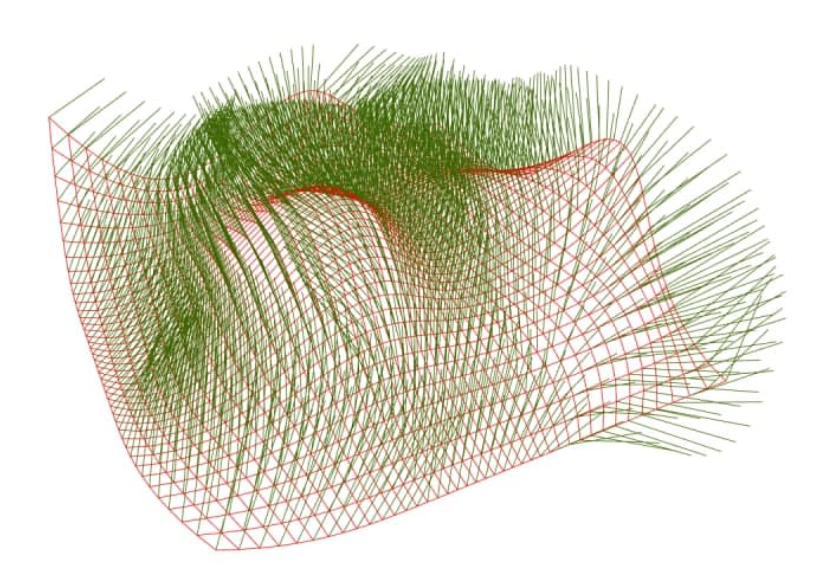


Рисунок 12 – Нормали к NURBS-поверхности

Сравнение характеристик алгоритмов

Произведем сравнение алгоритмов. Будем сравнивать такие характеристики, как время выполнения алгоритма и затраты памяти. Для тестирования алгоритмов использовалась NURBS-кривая третьей степени, задаваемая 5-ю контрольными точками, и NURBS-поверхность третьей степени, задающаяся сеткой контрольных точек . Время будем измерять в профилировщике chrom//tracing [16]. Сравнение алгоритмов представлено в таблице Таблица 1.

|  |  |  |  |
| --- | --- | --- | --- |
| Таблица 1 – Характеристики алгоритмов расчета NURBS-кривых и поверхностей и их первых производных | | | |
| Название алгоритма | Время выполнения для 40 тыс. точек, мс | Время выполнения для 1 млн. точек, мс | Затраты памяти, байты |
| Алгоритм расчета точки кривой с помощью базисных функций (Алг. 3) | 1.599 | 31.716 | 224 |
| Алгоритм Кокса – де Бура для расчета точки кривой (Алг. 4) | 1.892 | 39.039 | 208 |
| Алгоритм расчета производной в точке кривой (Алг. 6) | 3.015 | 57.315 | 496 |
| Алгоритм расчета точки поверхности с помощью базисных функций (Алг. 7) | 3.079 | 60.914 | 448 |
| Алгоритм Кокса – де Бура для расчета точки поверхности (Алг. 8) | 6.764 | 158.271 | 864 |
| Алгоритм расчета первых производных в точке поверхности (Алг. 9) | 18.393 | 166.510 | 1184 |

На основе полученных данных можно сделать вывод, что алгоритмы на основе базисных функций работают быстрее примерно на 15 – 20% для кривых и примерно на 55 – 60% быстрее для поверхностей, чем соответствующие алгоритмы Кокса – де Бура. Алгоритм Кокса – де Бура особенно неэффективен для поверхностей, поскольку хранит в буфере данные о точках. Алгоритмы вычисления производных показывают себя, как вполне эффективные.

ЗАКЛЮЧЕНИЕ

В ходе выполнения квалификационной работы студента было сделано следующее:

* произведен обзор теории алгоритмов расчета точек и производных B-сплайнов;
* выведены уравнения B-сплайна и его производных на основе алгоритма Кокса – де Бура и расчета базисных функций;
* на основе выведенных уравнений были составлены соответствующие алгоритмы;
* выведенные алгоритмы были обобщены на случай NURBS;
* с помощью разработанных алгоритмов были построены NURBS кривые и поверхности, а также с помощью расчета производных визуализированы касательные для кривых и нормали для поверхностей;
* проведено сравнение алгоритмов по времени выполнения и затратам памяти.

# **СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ**

1. Френсис Хилл «OpenGL. Программирование компьютерной графики»
2. Н. П. Корнейчук «Сплайны в теории приближения»
3. Jeffrey M. Lane, Richard F. Riesenfeld «A Theoretical Development for the Computer Generation and Display of Piecewise Polynomial Surfaces»
4. Kang Min, Yanyan Sun, Chen‑Han Lee, Pengcheng Hu, Shanshan He «An Improved B‑Spline Fitting Method with Arc‑Length Parameterization, G2‑continuous Blending, and Quality Refinement»
5. А. А. Захаров «Геометрическое моделирование» лекции
6. D. Salomon «Curves and Surfaces for Computer Graphics»
7. Б. И. Квасов «Методы изогеометрической аппроксимации сплайнами»
8. А. В. Приемышев, В. Н. Крутов «Компьютерная графика в САПР»
9. Кунву Ли. «Основы САПР CAD/CAM/CAE»
10. Карл де Бур «Практическое руководство по сплайнам»
11. Les Piegl, Wayne Tiller «The NURBS Book»
12. Rogers D.F. Introduction to NURBS. With Historical Perspective
13. В. Е. Турлапов Моделирование кривых и поверхностей. Сплайны
14. А.С. Кольцов Е.Д. Федорков «Геометрическое моделирование в САПР»
15. Ю. И. Димитриенко «Тензорное исчисление»
16. https://www.chromium.org/developers/how-tos/trace-event-profiling-tool

ПРИЛОЖЕНИЕ А Программная реализация алгоритмов

**Алгоритм 1:**

u64 NURBSCurveBuilder::FindSpan(f64 p\_t)

{

u64 n = m\_Curve->m\_ControlPoints.size() - 1;

u64 p = m\_Curve->m\_Degree;

auto& U = m\_Curve->m\_Knots;

if (p\_t == U[n + 1]) return n;

u64 low = p;

u64 high = n + 1;

u64 mid = (low + high) / 2;

while (p\_t < U[mid] || p\_t >= U[mid + 1]) {

if (p\_t < U[mid]) high = mid;

else low = mid;

mid = (low + high) / 2;

}

return mid;

}

**Алгоритм 2:**

void NURBSCurveBuilder::BasisFuns(u64 p\_i, f64 p\_t)

{

u64 p = m\_Curve->m\_Degree;

auto& U = m\_Curve->m\_Knots;

m\_N[0] = 1.0;

for (size\_t j = 1; j <= p; ++j) {

m\_Left[j] = p\_t - U[p\_i + 1 - j];

m\_Right[j] = U[p\_i + j] - p\_t;

f64 saved = 0.0;

for (size\_t r = 0; r < j; ++r) {

f64 temp = m\_N[r] / (m\_Right[r + 1] + m\_Left[j - r]);

m\_N[r] = saved + m\_Right[r + 1] \* temp;

saved = m\_Left[j - r] \* temp;

}

m\_N[j] = saved;

}

}

**Алгоритм 3:**

Vec3D NURBSCurveBuilder::CalculatePoint(f64 p\_t)

{

u64 n = m\_Curve->m\_ControlPoints.size() - 1;

u64 p = m\_Curve->m\_Degree;

auto& P = m\_Curve->m\_ControlPoints;

u64 span = FindSpan(p\_t);

BasisFuns(span, p\_t);

Vec3D o\_point;

f64 weight = 0;

for (size\_t i = 0; i <= p; ++i) {

o\_point.x += m\_N[i] \* P[span - p + i].x \* P[span - p + i].w;

o\_point.y += m\_N[i] \* P[span - p + i].y \* P[span - p + i].w;

o\_point.z += m\_N[i] \* P[span - p + i].z \* P[span - p + i].w;

weight += m\_N[i] \* P[span - p + i].w;

}

o\_point.x /= weight;

o\_point.y /= weight;

o\_point.z /= weight;

return o\_point;

}

**Алгоритм 4:**

Vec3D NURBSCurveBuilder2::CalculatePoint(f64 p\_t)

{

const u64 k = m\_Curve->m\_Degree + 1;

const auto& P = m\_Curve->m\_ControlPoints;

const auto& U = m\_Curve->m\_Knots;

const u64 l = FindSpan(p\_t);

for (size\_t j = 0; j < k; ++j) {

m\_PointBuffer[j] = P[l - k + 1 + j].xyz() \* P[l - k + 1 + j].w;

m\_WeightBuffer[j] = P[l - k + 1 + j].w;

}

for (size\_t r = 1; r < k; ++r) {

for (size\_t j = k - 1; j >= r; --j) {

size\_t i = l - k + 1 + j;

const f64 left = p\_t - U[i];

const f64 right = U[i + k - r] - p\_t;

m\_PointBuffer[j] = (m\_PointBuffer[j].scaled(left) + m\_PointBuffer[j - 1].scaled(right)).scaled(1.0 / (left + right));

m\_WeightBuffer[j] = (m\_WeightBuffer[j] \* left + m\_WeightBuffer[j - 1] \* right) / (left + right);

}

}

return m\_PointBuffer[k - 1].scaled(1.0 / m\_WeightBuffer[k - 1]);

}

**Алгоритм 5:**

void NURBSCurveBuilder::BasisFunsFirstDerivatives(u64 p\_i, f64 p\_t)

{

const u64 p = m\_Curve->m\_Degree;

const std::vector<f64>& U = m\_Curve->m\_Knots;

# define \_ndu(i, j) m\_ndu[(i) \* (p+1) + (j)]

# define \_ders(i, j) m\_Ders[(i) \* (p+1) + (j)]

# define \_a(i, j) m\_a[(i) \* 2 + (j)]

\_ndu(0, 0) = 1.0;

for (u64 j = 1; j <= p; ++j) {

m\_Left[j] = p\_t - U[p\_i + 1 - j];

m\_Right[j] = U[p\_i + j] - p\_t;

f64 saved = 0.0;

for (u64 r = 0; r < j; ++r) {

\_ndu(j, r) = m\_Right[r + 1] + m\_Left[j - r];

const f64 temp = \_ndu(r, j - 1) / \_ndu(j, r);

\_ndu(r, j) = saved + m\_Right[r + 1] \* temp;

saved = m\_Left[j - r] \* temp;

}

\_ndu(j, j) = saved;

}

//..................................................

// Loading the basis functions

for (i64 j = 0; j <= p; ++j) {

\_ders(0, j) = \_ndu(j, p);

}

//..................................................

// This section computes the derivatives

for (i64 r = 0; r <= p; ++r) {

i64 s1 = 0;

i64 s2 = 1;

\_a(0, 0) = 1.0;

f64 d = 0.0;

i64 r1 = r - 1;

i64 p1 = p - 1;

if (r >= 1) {

\_a(s2, 0) = \_a(s1, 0) / \_ndu(p1 + 1, r1);

d = \_a(s2, 0) \* \_ndu(r1, p1);

}

i64 j1, j2;

if (r1 >= -1) j1 = 1;

else j1 = -r1;

if (r - 1 <= p1) j2 = 0;

else j2 = p - r;

for (i64 j = j1; j <= j2; ++j) {

\_a(s2, j) = (\_a(s1, j) - \_a(s1, j - 1)) / \_ndu(p1 + 1, r1 + j);

d += \_a(s2, j) \* \_ndu(r1 + j, p1);

}

if (r <= p1) {

\_a(s2, 1) = -\_a(s1, 0) / \_ndu(p1 + 1, r);

d += \_a(s2, 1) \* \_ndu(r, p1);

}

\_ders(1, r) = d;

const i64 tmp = s1;

s1 = s2;

s2 = tmp;

}

//..................................................

// Multiply through by the correct factors

u64 r = p;

for (u64 j = 0; j <= p; ++j) {

\_ders(1, j) \*= r;

}

}

# undef \_ndu

# undef \_ders

# undef \_a

}

**Алгоритм 6:**

Vec3D NURBSCurveBuilder::CalculateFirstDerivative(f64 p\_t)

{

const u64 p = m\_Curve->m\_Degree;

const auto& U = m\_Curve->m\_Knots;

const auto& P = m\_Curve->m\_ControlPoints;

const u64 n = m\_Curve->m\_ControlPoints.size() - 1;

u64 span = FindSpan(p\_t);

BasisFunsFirstDerivatives(span, p\_t);

# define \_ders(i, j) m\_Ders[(i) \* (p+1) + (j)]

Vec4D C1 = { 0, 0, 0, 0 };

Vec4D C0 = { 0, 0, 0, 0 };

for (u64 j = 0; j <= p; ++j) {

C0.x += \_ders(0, j) \* P[span - p + j].x \* P[span - p + j].w;

C0.y += \_ders(0, j) \* P[span - p + j].y \* P[span - p + j].w;

C0.z += \_ders(0, j) \* P[span - p + j].z \* P[span - p + j].w;

C0.w += \_ders(0, j) \* P[span - p + j].w;

C1.x += \_ders(1, j) \* P[span - p + j].x \* P[span - p + j].w;

C1.y += \_ders(1, j) \* P[span - p + j].y \* P[span - p + j].w;

C1.z += \_ders(1, j) \* P[span - p + j].z \* P[span - p + j].w;

C1.w += \_ders(1, j) \* P[span - p + j].w;

}

Vec3D o\_derivative;

o\_derivative.x = (C0.w \* C1.x - C1.w \* C0.x) / (C0.w \* C0.w);

o\_derivative.y = (C0.w \* C1.y - C1.w \* C0.y) / (C0.w \* C0.w);

o\_derivative.z = (C0.w \* C1.z - C1.w \* C0.z) / (C0.w \* C0.w);

return o\_derivative;

# undef \_ders

}

**Алгоритм 7:**

Vec3D NURBSSurfaceBuilder::CalculatePoint(f64 p\_t, f64 p\_s)

{

u64 n = m\_Surface->m\_tControlPointsCount - 1;

u64 pT = m\_Surface->m\_DegreeT;

u64 m = m\_Surface->m\_sControlPointsCount - 1;

u64 pS = m\_Surface->m\_DegreeS;

auto& P = m\_Surface->m\_ControlPoints;

auto spanTS = FindSpan(p\_t, p\_s);

BasisFuns(spanTS.first, p\_t, spanTS.second, p\_s);

Vec3D S;

f64 weight = 0;

for (size\_t i = 0; i <= pT; ++i) {

for (size\_t j = 0; j <= pS; ++j) {

S.x += m\_NT[i] \* m\_NS[j] \* P[(spanTS.first - pT + i) \* (m + 1) + (spanTS.second - pS + j)].x \* P[(spanTS.first - pT + i) \* (m + 1) + (spanTS.second - pS + j)].w;

S.y += m\_NT[i] \* m\_NS[j] \* P[(spanTS.first - pT + i) \* (m + 1) + (spanTS.second - pS + j)].y \* P[(spanTS.first - pT + i) \* (m + 1) + (spanTS.second - pS + j)].w;

S.z += m\_NT[i] \* m\_NS[j] \* P[(spanTS.first - pT + i) \* (m + 1) + (spanTS.second - pS + j)].z \* P[(spanTS.first - pT + i) \* (m + 1) + (spanTS.second - pS + j)].w;

weight += m\_NT[i] \* m\_NS[j] \* P[(spanTS.first - pT + i) \* (m + 1) + (spanTS.second - pS + j)].w;

}

}

S.x /= weight;

S.y /= weight;

S.z /= weight;

return S;

}

**Алгоритм 8:**

Vec3D NURBSSurfaceBuilder2::CalculatePoint(f64 p\_t, f64 p\_s)

{

f64 left, right;

u64 kT = m\_Surface->m\_DegreeT + 1;

u64 kS = m\_Surface->m\_DegreeS + 1;

u64 n = m\_Surface->m\_tControlPointsCount - 1;

u64 m = m\_Surface->m\_sControlPointsCount - 1;

auto& P = m\_Surface->m\_ControlPoints;

const auto& U = m\_Surface->m\_KnotsT;

const auto& V = m\_Surface->m\_KnotsS;

auto spanTS = FindSpan(p\_t, p\_s);

# define \_P(i, j) P[(i) \* (m+1) + (j)]

# define \_PB(i, j) m\_PointBuffer[(i) \* kS + (j)]

# define \_WB(i, j) m\_WeightBuffer[(i) \* kS + (j)]

for (size\_t i = 0; i < kT; ++i) {

for (size\_t j = 0; j < kS; ++j) {

\_PB(i, j) = \_P(spanTS.first - kT + 1 + i, spanTS.second - kS + 1 + j).xyz().scaled(\_P(spanTS.first - kT + 1 + i, spanTS.second - kS + 1 + j).w);

\_WB(i, j) = \_P(spanTS.first - kT + 1 + i, spanTS.second - kS + 1 + j).w;

}

}

for (size\_t j = 0; j < kS; ++j) {

for (size\_t r = 1; r < kT; ++r) {

for (size\_t i = kT - 1; i >= r; --i) {

const u64 k = spanTS.first - kT + 1 + i;

const f64 left = p\_t - U[k];

const f64 right = U[k + kT - r] - p\_t;

\_PB(i, j) = (\_PB(i, j).scaled(left) + \_PB(i - 1, j).scaled(right)).scaled(1.0 / (left + right));

\_WB(i, j) = (\_WB(i, j) \* left + \_WB(i - 1, j) \* right) / (left + right);

}

}

}

for (size\_t r = 1; r < kS; ++r) {

for (size\_t j = kS - 1; j >= r; --j) {

const u64 k = spanTS.second - kS + 1 + j;

const f64 left = p\_s - V[k];

const f64 right = V[k + kS - r] - p\_s;

\_PB(kT - 1, j) = (\_PB(kT - 1, j).scaled(left) + \_PB(kT - 1, j - 1).scaled(right)).scaled(1.0 / (left + right));

\_WB(kT - 1, j) = (\_WB(kT - 1, j) \* left + \_WB(kT - 1, j - 1) \* right) / (left + right);

}

}

return \_PB(kT - 1, kS - 1).scaled(1.0 / \_WB(kT - 1, kS - 1));

# undef \_P

# undef \_PB

# undef \_WB

}

**Алгоритм 9:**

std::pair<Vec3D, Vec3D> NURBSSurfaceBuilder::CalculateFirstDerivative(f64 p\_t, f64 p\_s)

{

const u64 pT = m\_Surface->m\_DegreeT;

const u64 pS = m\_Surface->m\_DegreeS;

const std::vector<f64>& UT = m\_Surface->m\_KnotsT;

const std::vector<f64>& US = m\_Surface->m\_KnotsS;

const auto& P = m\_Surface->m\_ControlPoints;

const u64 n = m\_Surface->m\_tControlPointsCount - 1;

const u64 m = m\_Surface->m\_sControlPointsCount - 1;

auto spanTS = FindSpan(p\_t, p\_s);

BasisFunsFirstDerivatives(spanTS.first, p\_t, spanTS.second, p\_s);

# define \_dersT(i, j) m\_DersT[(i) \* (pT+1) + (j)]

# define \_dersS(i, j) m\_DersS[(i) \* (pS+1) + (j)]

Vec4D C0 = { 0, 0, 0, 0 };

for (size\_t i = 0; i <= pT; ++i) {

for (size\_t j = 0; j <= pS; ++j) {

C0.x += \_dersT(0, i) \* \_dersS(0, j) \* P[(spanTS.first - pT + i) \* (m + 1) + (spanTS.second - pS + j)].x \* P[(spanTS.first - pT + i) \* (m + 1) + (spanTS.second - pS + j)].w;

C0.y += \_dersT(0, i) \* \_dersS(0, j) \* P[(spanTS.first - pT + i) \* (m + 1) + (spanTS.second - pS + j)].y \* P[(spanTS.first - pT + i) \* (m + 1) + (spanTS.second - pS + j)].w;

C0.z += \_dersT(0, i) \* \_dersS(0, j) \* P[(spanTS.first - pT + i) \* (m + 1) + (spanTS.second - pS + j)].z \* P[(spanTS.first - pT + i) \* (m + 1) + (spanTS.second - pS + j)].w;

C0.w += \_dersT(0, i) \* \_dersS(0, j) \* P[(spanTS.first - pT + i) \* (m + 1) + (spanTS.second - pS + j)].w;

}

}

Vec4D C1T = { 0, 0, 0, 0 };

for (size\_t i = 0; i <= pT; ++i) {

for (size\_t j = 0; j <= pS; ++j) {

C1T.x += \_dersT(1, i) \* \_dersS(0, j) \* P[(spanTS.first - pT + i) \* (m + 1) + (spanTS.second - pS + j)].x \* P[(spanTS.first - pT + i) \* (m + 1) + (spanTS.second - pS + j)].w;

C1T.y += \_dersT(1, i) \* \_dersS(0, j) \* P[(spanTS.first - pT + i) \* (m + 1) + (spanTS.second - pS + j)].y \* P[(spanTS.first - pT + i) \* (m + 1) + (spanTS.second - pS + j)].w;

C1T.z += \_dersT(1, i) \* \_dersS(0, j) \* P[(spanTS.first - pT + i) \* (m + 1) + (spanTS.second - pS + j)].z \* P[(spanTS.first - pT + i) \* (m + 1) + (spanTS.second - pS + j)].w;

C1T.w += \_dersT(1, i) \* \_dersS(0, j) \* P[(spanTS.first - pT + i) \* (m + 1) + (spanTS.second - pS + j)].w;

}

}

Vec4D C1S = { 0, 0, 0, 0 };

for (size\_t i = 0; i <= pT; ++i) {

for (size\_t j = 0; j <= pS; ++j) {

C1S.x += \_dersT(0, i) \* \_dersS(1, j) \* P[(spanTS.first - pT + i) \* (m + 1) + (spanTS.second - pS + j)].x \* P[(spanTS.first - pT + i) \* (m + 1) + (spanTS.second - pS + j)].w;

C1S.y += \_dersT(0, i) \* \_dersS(1, j) \* P[(spanTS.first - pT + i) \* (m + 1) + (spanTS.second - pS + j)].y \* P[(spanTS.first - pT + i) \* (m + 1) + (spanTS.second - pS + j)].w;

C1S.z += \_dersT(0, i) \* \_dersS(1, j) \* P[(spanTS.first - pT + i) \* (m + 1) + (spanTS.second - pS + j)].z \* P[(spanTS.first - pT + i) \* (m + 1) + (spanTS.second - pS + j)].w;

C1S.w += \_dersT(0, i) \* \_dersS(1, j) \* P[(spanTS.first - pT + i) \* (m + 1) + (spanTS.second - pS + j)].w;

}

}

Vec3D o\_derivativeT;

o\_derivativeT.x = (C0.w \* C1T.x - C1T.w \* C0.x) / (C0.w \* C0.w);

o\_derivativeT.y = (C0.w \* C1T.y - C1T.w \* C0.y) / (C0.w \* C0.w);

o\_derivativeT.z = (C0.w \* C1T.z - C1T.w \* C0.z) / (C0.w \* C0.w);

Vec3D o\_derivativeS;

o\_derivativeS.x = (C0.w \* C1S.x - C1S.w \* C0.x) / (C0.w \* C0.w);

o\_derivativeS.y = (C0.w \* C1S.y - C1S.w \* C0.y) / (C0.w \* C0.w);

o\_derivativeS.z = (C0.w \* C1S.z - C1S.w \* C0.z) / (C0.w \* C0.w);

return std::make\_pair(o\_derivativeT, o\_derivativeS);

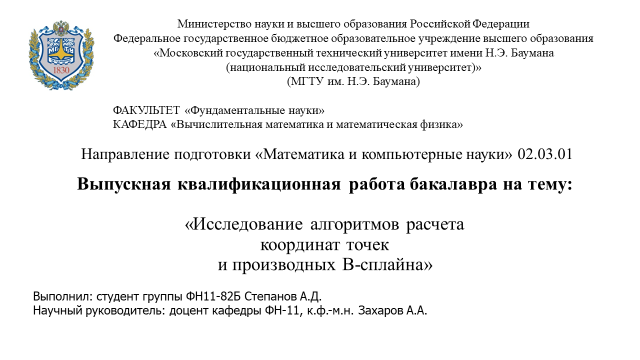
# undef \_dersT

# undef \_dersS

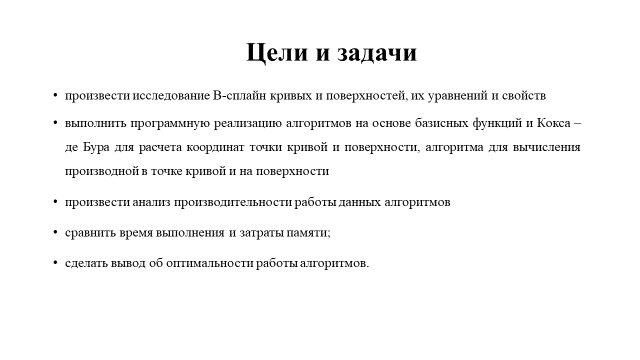
}

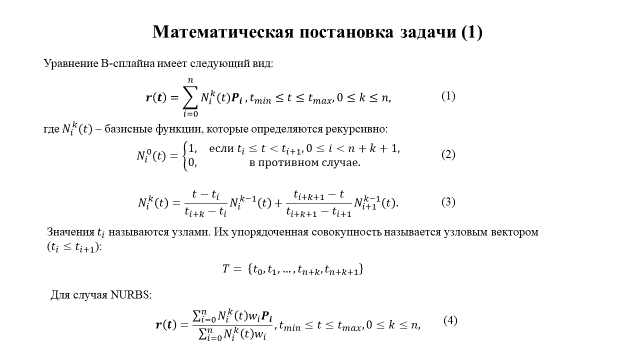
ПРИЛОЖЕНИЕ Б Графическая часть ВКР

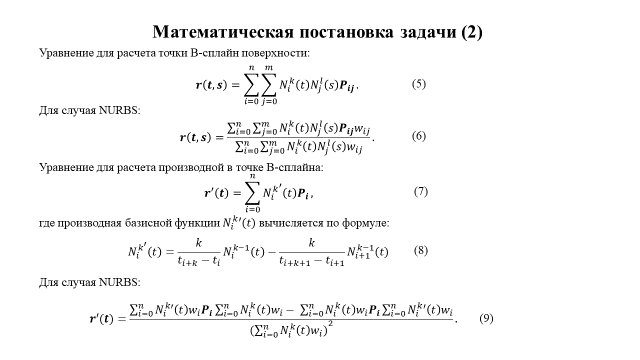
В графическую часть выпускной квалификационной работы входят слайды презентации в количестве 26 шт.

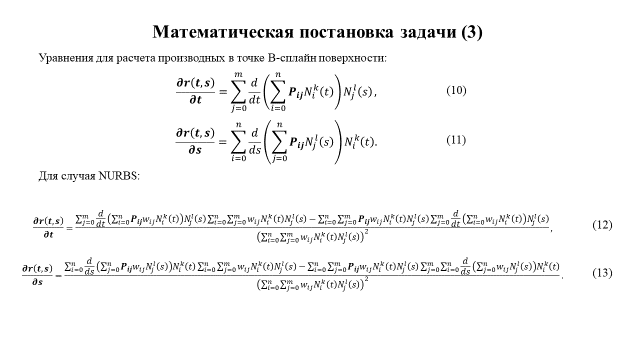












26