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

ФАКУЛЬТЕТ **ИНФОРМАТИКА И СИСТЕМЫ УПРАВЛЕНИЯ**

КАФЕДРА **КОМПЬЮТЕРНЫЕ СИСТЕМЫ И СЕТИ (ИУ6)**

НАПРАВЛЕНИЕ ПОДГОТОВКИ **09.03.01 ИНФОРМАТИКА И ВЫЧИСЛИТЕЛЬНАЯ ТЕХНИКА**

О Т Ч Е Т

по лабораторной работе № 1

Название: Построение семантической сети документов с помощью программы Complearn

Дисциплина: Методы и алгоритмы сжатия данных

Вариант 10

Студент ИУ6-63Б _____ В.К. Залыгин
(Группа) (Подпись, дата) (И.О. Фамилия)

Преподаватель _____ В.М.Григоренко
(Подпись, дата) (И.О. Фамилия)

Москва, 2025

Цель работы

Знакомство с основными принципами построения семантической сети. Освоение навыков анализа семантической сети документов.

Ход работы:

Изначально было решено сравнить тексты, заданные преподавателем. По варианту, номера текстов – 1 и 3 (10=4+4+2 вариант). Перейдем по ссылкам и скопируем текст страниц в 2 файла соответственно, которые необходимо поместить в директорию с программой.

После этого, запустим программу для нахождения NCD для 2-х текстов, выданных преподавателем. Выполненная команда и ее результат показаны на рисунке 1.

```
PS C:\repos\bmstu-ics6\zhim-zhim\imba> ./ncd 1.txt 3.txt
 0,936570
PS C:\repos\bmstu-ics6\zhim-zhim\imba>
```

Рисунок 1 – Результат работы команды ncd для текстов преподавателя

Далее необходимо выбрать 2 текста самостоятельно, и также как и до этого – сравнить их. В качестве первого текста выбрана статья https://habr.com/ru/articles/903016/. В качестве второго -- https://habr.com/ru/companies/ruvds/articles/901944/. Результат выполнения команды показан на рисунке 2.

```
PS C:\repos\bmstu-ics6\zhim-zhim\imba> ./ncd 2.txt 4.txt
 0,962923
PS C:\repos\bmstu-ics6\zhim-zhim\imba>
```

Рисунок 2 – Результат работы команды ncd для текстов студента

После этого проведем еще 4 сравнения, таких, чтоб все тексты были попарно сравнены. Результат сравнений показан на рисунке 3.

```
PS C:\repos\bmstu-ics6\zhim-zhim\imba> ./ncd 1.txt 2.txt
 0,949266
PS C:\repos\bmstu-ics6\zhim-zhim\imba> ./ncd 2.txt 3.txt
 0,969058
PS C:\repos\bmstu-ics6\zhim-zhim\imba> ./ncd 3.txt 4.txt
 0,937696
PS C:\repos\bmstu-ics6\zhim-zhim\imba> ./ncd 1.txt 4.txt
 0,941039
PS C:\repos\bmstu-ics6\zhim-zhim\imba>
```

Рисунок 3 – Оставшиеся попарные сравнения

Затем с помощью команды maketree для файла с ключевыми словами для одного из текстов сгенерирован файл с деревом расширения .dot. Полученный граф изображен на рисунках 4 и 5.

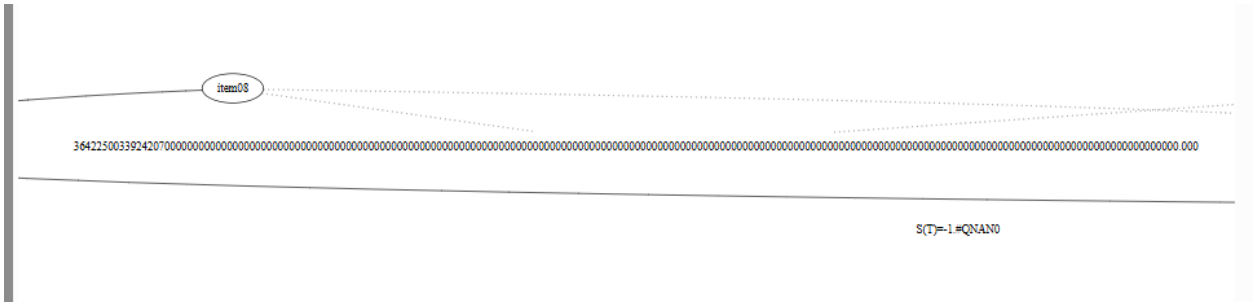

Рисунок 4 — Первая часть сгенерированного дерева

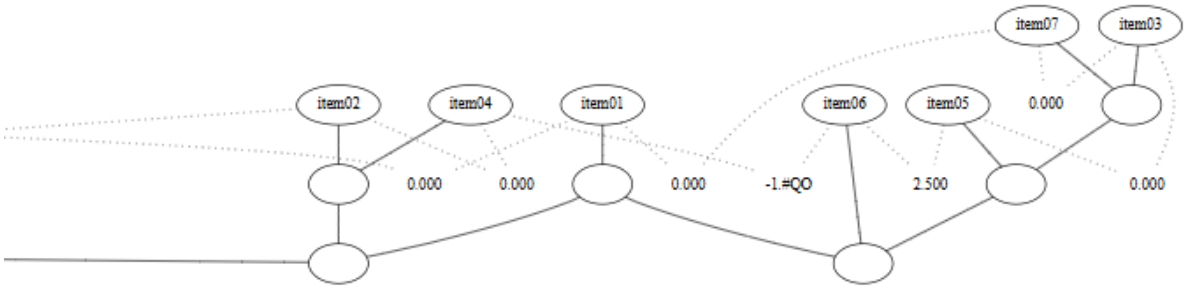

Рисунок 5 — Вторая часть сгенерированного дерева

Содержимое .dot-файла приведено в листинге 1.

Листинг 1 – Содержимое .dot-файла

```
graph "tree" {
label="S(T)=-1.#QNAN0";
1 [label="item01"];
6 [label="item02"];
3 [label="item03"];
5 [label="item04"];
0 [label="item05"];
7 [label="item06"];
4 [label="item07"];
2 [label="item08"];
8 [label=""];
9 [label=""];
10 [label=""];
11 [label=""];
12 [label=""];
13 [label=""];
```

```
1 -- 8 [weight="2"];
6 -- 10 [weight="2"];
3 -- 9 [weight="2"];
5 -- 10 [weight="2"];
0 -- 11 [weight="2"];
7 -- 13 [weight="2"];
4 -- 9 [weight="2"];
2 -- 12 [weight="2"];
8 -- 12 [weight="2"];
8 -- 13 [weight="2"];
9 -- 11 [weight="2"];
10 -- 12 [weight="2"];
11 -- 13 [weight="2"];
1 -- 10014 [style="dotted"];
4 -- 10014 [style="dotted"];
10014 [label="0.000",color="white"];
4 -- 10015 [style="dotted"];
3 -- 10015 [style="dotted"];
10015 [label="0.000",color="white"];
3 -- 10016 [style="dotted"];
0 -- 10016 [style="dotted"];
10016 [label="0.000",color="white"];
0 -- 10017 [style="dotted"];
7 -- 10017 [style="dotted"];
10017 [label="2.500",color="white"];
7 -- 10018 [style="dotted"];
5 -- 10018 [style="dotted"];
10018 [label="-1.#QO",color="white"];
5 -- 10019 [style="dotted"];
6 -- 10019 [style="dotted"];
10019 [label="0.000",color="white"];
6 -- 10020 [style="dotted"];
2 -- 10020 [style="dotted"];
10020 [la-
bel="364225003392420700000000000000000000000000000000000
0000000000000000000000000000000000000000000000000000
0000000000000000000000000000000000000000000000000000
00000000000.000",color="white"];
2 -- 10021 [style="dotted"];
1 -- 10021 [style="dotted"];
10021 [label="0.000",color="white"];
}
```

Вывод: в результате выполнения лабораторной работы была изучена программа Complearn для применения техник сжатия. Были применены основные возможности данной программы для нахождения нормализованного расстояния сжатия.

ПРИЛОЖЕНИЕ А

Текст 1 (выбран преподавателем):

Обучать профессионалов намного сложнее, чем новичков. Если новичок – это чистый лист, то профи – сам как живой учебник, наполненный знаниями, устоявшимися взглядами и интересами. Но даже профессионалу иной раз нужна помощь в достижении целей – для этого мы и создали Data Science Акселератор. Подробнее о том, что это такое, из чего состоит и как создавалось – под катом.

Что это такое?

Data Science Акселератор – это трёхмесячный интенсив, созданный для специалистов с опытом (или хотя бы окончивших курс Data Science) и нуждающихся в практике проработки одного проекта. Программа Акселератора составлена таким образом, чтобы помочь достичь трёх больших целей:

закрыть пробелы в знаниях и изучить материалы;

реализовать личный pet-project, проект для своего бизнеса или конкретную рабочую задачу под пристальным наблюдением менторов;

выявить и устранить сложности с устройством на работу (для тех, у кого они есть).

Первая группа студентов-участников Акселератора получилась довольно разношёрстная и интересная: в группе был дата-сайентист из банка, желающий реализовать проект по сбору промокодов из видео, текстов, изображений. Также в Акселератор пришел дата-сайентист @vandriichuk, разрабатывающий рекомендательную систему для интернет-магазина. Был у нас и владелец интернет-магазина @Walker2000 создающий умных ботов на

основе нейросетей для e-commerce. Также к нам попал сотрудник Роботолаб Сколково, использующий технологии NLP для своего робота, и многие другие. У каждого – свой бэкграунд, своя цель. Кто-то хотел сменить работу, кто-то – получить знания от топ-экспертов, кто-то работает над своим собственным проектом и подумывает превратить его в стартап.

Список проектов студентов, с которыми они пришли в Акселератор, тоже приятно удивляет (сколько всё-таки талантливых людей вокруг!):

рекомендательная система для интернет-магазина;

саммаризация текстов (статьи, научные публикации и т. д.); Несколько более упрощённый вариант – выделение тем в коллекции документов;

Instagram-бот коммерческого аккаунта для поиска, оценки и взаимодействия с инфлюенсерами;

алгоритм для оценки эмоциональной тональности сообщений, сгенерированных GPT-3 на заданную тему;

извлечение промокодов из контента в соцсетях;

робот-предсказатель;

прогнозирование задержек авиарейсов;

анализ видео из беспилотных автомобилей;

создание предиктивной модели нетарифных барьеров в торговле молочной продукцией и расширение её до 6 видов товаров животного и растительного происхождения.

Из чего состоит Акселератор

Главный вопрос, который стоял перед создателями Акселератора, – как сделать обучение эффективным для каждого студента? Ведь нужно учесть бэкграунд каждого, их склонности в работе и даже личные особенности. Подобных вещей в образовании data science ранее никто не делал (ни в России, ни за рубежом), и поэтому нам пришлось делать всё самим с нуля. В этом был настоящий драйв: те, кто делал свои продукты, нас поймут. Вот из чего в итоге состояла "начинка" Акселератора:

Постановка персональных целей – нужна для того, чтобы сделать обучение максимально полезным для студента.

Персональные траектории обучения – мы сделали программы по NLP, ML и CV, пригласив работать над ним экспертов из Google, Яндекса, Mail.ru, Сбера и М-видео.

Менторы из топ-экспертов сферы, привлечённые к работе над персональным проектом или профессиональной задачей каждого участника. Акселератор оказался интересным для экспертов самого высокого уровня, потому что сами студенты были не новички, и над проектами они работали непростыми. К нам присоединились Валерий Бабушкин из Facebook, Валентин Малых из Huawei, Эмиль Магеррамов из Biocad и другие.

Еженедельные встречи с трекингом поставленных целей и работой над контентом. Студенты учились тому, как, где и кому рассказывать о своих

проектах, как создавать сообщество вокруг себя, выступали в качестве спикеров на профильных конференциях.

Специальный карьерный блок, включающий в себя вебинары и тренинги по прохождению собеседований и встречи один на один с карьерными консультантами.

Дальше расскажем про каждый пункт подробнее.

1. Постановка персональных целей
Чтобы будущий участник не просто ставил цель по реализации проекта или обучению, но и связывал своё обучение с профессиональным будущим, своими стратегическими планами, ожиданиями, мы сформировали специальный опросник.

Пример вопросов анкеты:

Топ 3-компании, в которых я бы хотел работать (если вы пока не знаете, напишите ”пока не определился”).

Позиция, на которой я бы хотел работать после прохождения акселератора.

Сфера деятельности (биотех, коммерция, авиация, образование), в которой вы бы хотели работать в дальнейшем (если вы не определились или это не имеет значения, так и напишите).

Этот опросник также помог подобрать лучших менторов и экспертов непосредственно под наших студентов, подчас даже из тех компаний, куда хотели попасть студенты.

2. Индивидуальные траектории обучения

В основе программы Акселератора – несколько разных курсов. Например, один из них – собранный интенсивный курс по NLP, ML, CV с практическими кейсами вместо упражнений. С каждым студентом мы выбирали базовый курс и после этого изменяли его в соответствии с персональными запросами, задачами и бэкграундом студента. Также мы дополнительно включили доступ к темам, которые являются основополагающими, чтобы при необходимости участники могли подтянуть базу и закрыть пробелы в знаниях. Так каждый получил свою индивидуальную траекторию, которую процессе работы над проектом студент и ментор могли корректировать уже под конкретный проект.

Отдельное упражнение для студентов до старта – просмотреть собранную под их задачи программу – так формировался первый вариант, который мы утверждали сперва со студентом, а потом с ментором.

Консультация по личной индивидуальной траектории – это не только договорённость о том, что именно будет содержать программа, но и выявление потенциальных сложных моментов на старте обучения (были случаи, когда участники меняли программу в ходе таких консультаций).

Итого: траектория строилась на одном из продвинутых курсов по запросу участника, затем гибко подстраивалась под уровень знаний участника в каждой теме, а затем затачивалась под проект (на старте и в процессе).

3. Менторы и программные эксперты

Главная функция менторов в Акселераторе – двигать участников в работе над проектами. Именно они отслеживают прогресс и дают обратную связь, встречаются каждую неделю с участниками. Ментор – это ключевое звено, нужное, чтобы проект участника был реализован. Это как со спортзалом: если человек просто купил абонемент, то он может и не ходить. Но после покупки

персональной тренировки человек с большей вероятностью пойдёт в зал. Потому что люди предпочитают учиться у конкретных людей, у кого считают интересным учиться.

Ментор – не просто человек, который показывает, что ты сделал не так, это твой наставник, который помогает искать решения. Кроме того, есть неформальные знания, которые передаются, только если вы работаете непосредственно с людьми в одной команде. Поэтому мы старались подбирать менторов, которые работают в той же сфере и даже в той компании, где хотел бы работать участник Акселератора.

Эксперты, задействованные в разработке программе, – признанные профессионалы отрасли, уже достигшие того уровня, к которому стремятся наши студенты. Акселератор давал участникам целевое окружение, где с каждым экспертом имелась возможность обсудить собственный проект.

Обсуждение профессиональных тем с лучшими экспертами области – одна из тех возможностей, которую часто недооценивают. Мы понимаем, как это важно, и строим программу на коммуникациях, стремимся интегрировать участников в профессиональное сообщество.

На каждой защите проекта студенты встречались с разными экспертами, которые с разных позиций рассматривали проект, давали конкретные оценки и рекомендации.

4. Коммуникации в Акселераторе, или "социалка", которой нет

“Социалкой”, или социальными, групповыми механиками, называются те механики, которые требуют взаимодействия от участников. Как правило, они помогают осваивать материал, включаться в обучение. При использовании таких механик повышается успешное прохождение в онлайн-курсах.

Мы очень долго внутри спорили, что делать с "социалкой" при таком подходе. Казалось бы, создав персональные траектории, мы потеряли группу и то, что часто помогает участникам образовательных курсов, а также снижает нагрузку на специалистов поддержки курса.

Одна из важных механик в Акселераторе – это групповые встречи с менторами. Эти встречи нужны для расширения представления участников о задачах и подходах к решению, а также возможности консультирования студентов друг другом. Групповые встречи как один из моментов социального давления (договорённости о целях на неделю) также работают хорошо и позволяют держать участников в тонусе.

Итак, если у вас нет группы в её классическом, школьном понимании, это ещё не значит, что у вас нет группы. Задачи коммуникации, такие как проектирование групповых механик, могут отличаться от привычных нам и работать не менее эффективно на образовательный опыт участников.

Что происходит сейчас: участники консультируют друг друга, включаются во внешние профессиональные сообщества и выступают на конференциях. Один из участников Акселератора Георгий Брегман создал своё мини-сообщество, посвящённое нейронным сетям AI и ML — DSMLAI.

5. Карьерный блок

Обязательная часть программы. Некоторые из студентов уже проходили какие-то курсы и хотят устроиться на определенную работу, но по какой-то причине не могут. Именно здесь мы анализировали, почему это происходит, работали с резюме и позиционированием, проводили тренинги по собеседованиям.

В карьерный блок включались не только общие мероприятия, но и индивидуальные встречи с карьерными консультантами. Сочетание развития хард-скилов на практике, софт-скилов при работе с людьми над проектами и умение преподнести свой опыт дают в итоге кумулятивный эффект.

Карьерный блок позволил студентам оценить свои возможности, выделить точки роста и построить путь к той профессиональной позиции, которую участники себе ставили. В нём мы формируем привычку коммуницировать с работодателем, оценивать собственный опыт и соотносить его с запросами рынка.

Секретный ингредиент

«В зависимости от аудитории и программы, для того, чтобы добиваться эффекта мы используем разные секретные ингредиенты. В программе Акселератора, таким секретным ингредиентом является блок “Профессиональный бренд”. Как и карьерный блок, он выстраивается вокруг профессиональной цели. Кроме того, именно этот блок стал своеобразной коучинговой группой для наших студентов. Здесь они могли сказать, что не смогли сказать в обсуждении с ментором, искать ответы совместно.»

Марина Щербакова, педагогический дизайнер, разработчик образовательного продукта.

Не все студенты посещали этот блок, потому что не осознавали его ценность: казалось бы, зачем дата-сайентистам контент? На самом деле блок решал сразу несколько задач.

На первой встрече каждый студент ответил для себя на вопросы: Какой цели он хотел бы достичь? Как проект, над которым он работает, поможет ему достичь его цели? Кто может помочь ему в достижении его цели? Как и где эти

люди могут узнать о том, над каким проектом студент работает, и заинтересоваться им?

На каждой следующей встрече студент сверялся с трекингом – приблизился ли он к своей цели? Интересна ли ему эта цель по-прежнему или стоит её поменять? Логично, что студенты, у которых цель была сформулирована чётко и конкретно, довольно быстро поняли, как её достичь. Каждый для этого использовал свой собственный путь: кто-то создал телеграм-канал и целенаправленно стал его развивать, наполняя ценной информацией о новостях в AI и ML, вовлекая экспертов. Кто-то стал писать статьи на Хабре (например про Применение предобученной модели VGG16 для рекомендаций и Применение нейросети в Instagram), кто-то – писать в Фейсбуке, а кто-то комфортнее чувствует себя в Linkedin. Студенты отметили, что стали получать благодаря этому гораздо больше приглашений на собеседования, предложений сотрудничества, возможности участвовать в интересных им проектах.

Студенты помогали друг другу продумывать контент-план, подкидывали интересные идеи, комментировали посты, подружились и превратились в мощную группу поддержки друг для друга. Поскольку все из одной сферы, то советы часто оказывались очень ценными.

Этот контентный блок вела Екатерина Артюгина, маркетолог с опытом работы в Яндекс, HeadHunter и Mail.ru. Участники отметили, что неожиданно для них этот блок оказался одним из самых ценных.

Что дал Акселератор участникам

В пилотном потоке Акселератора было 10 человек. За почти 3 месяца программы трое студентов получили офферы в международные компании, один студент прошёл сертификацию AWS, двое – получили прибыль от увеличения клиентов и продаж в собственном бизнесе. Остальные четверо

планируют продолжать работу над своими проектами и выйти на их защиту через два-три месяца.

Так, и что дальше?

Многое из того, что удалось реализовать в первом Акселераторе Data Science, скоро появится в курсе "Профессия Data Scientist", в котором студенты будут не только осваивать NLP, ML и CV, но и решать реальные кейсы, аналогичные бизнес-задачам, под руководством опытных менторов. Кого привлекает сфера Data Science и кто хочет получить актуальную ещё многие годы специальность – добро пожаловать! Ну, а мы позаботимся о том, чтобы у вас получилось дойти до финала. Будет сложно, что уж скрывать, но интересно.

Текст 2 (выбран преподавателем):

Фракталы — удивительные математические объекты, подкупающие своей простотой и богатыми возможностями по построению объектов сложной природы при помощи всего лишь нескольких коэффициентов и простой итеративной схемы.

Именно эти возможности и позволяют использовать их для сжатия изображений, особенно для фотографий природы и прочих сложных самоподобных изображений.

В этой статье я постараюсь коротко дать ответ на простой вопрос: «Как же это делается?».

Для начала все-таки придется немного углубиться в математику и ввести несколько определений.

Нам пригодятся следующие:

Метрика — функция, заданная на пространстве, возвращающая расстояние между двумя точками этого пространства. Например, привычная нам Эвклидова метрика. Если на пространстве задана метрика, оно называется метрическим. Метрика должна удовлетворять определенным условиям, но не будем углубляться.

Сжимающее отображение (преобразование) — функция на метрическом пространстве, равномерно уменьшающая расстояние между двумя точками пространства. Например, y=0.5x.

Сжимающие отображения обладают важным свойством. Если взять любую точку и начать итеративно применять к ней одно и то же сжимающее отображение: f(f(f...f(x))), то результатом будет всегда одна и та же точка. Чем больше раз применим, тем точнее найдем эту точку. Называется она неподвижной точкой и для каждого сжимающего отображения она существует, причем только одна.

Несколько аффинных сжимающих отображений образуют систему итерированных функций (СИФ). По сути, СИФ — это множительная машина. Она принимает исходное изображение, искажает его, перемещает, и так несколько раз.

Например, вот так при помощи СИФ из трех функций строится треугольник Серпинского:

Исходный треугольник три раза множится, уменьшается и переносится. И так далее. Если так продолжать до бесконечности, получим известный фрактал площадью 0 и размерностью 1,585.

Если функции, входящие в СИФ,— сжимающие, то сама СИФ тоже имеет неподвижную точку. Вот только эта «точка» будет уже не привычной нам точкой в N-мерном пространстве, а множеством таких точек, то есть изображением. Оно называется аттрактором СИФ. Для СИФ, приведенной выше, аттрактором будет треугольник Серпинского.

Тут мы переходим на следующий уровень — в пространство изображений. В этом пространстве:

Точка пространства — это изображение.

Расстояние между точками показывает, насколько похожи изображения между собой, насколько «близки» (естественно, если его задать соответствующим образом).

Сжимающее отображение делает два любых изображения более похожими (в смысле заданной метрики).

Имея СИФ, найти аттрактор просто. Во всяком случае, имея под рукой компьютер. Можно взять абсолютно любое начальное изображение и начать применять к нему СИФ. Пример с тем же треугольником, но уже построенным из квадрата:

Получается, что для построения довольно сложной фигуры нам потребовалось 18 коэффициентов. Выигрыш, как говорится, налицо.

Вот если бы мы умели решать обратную задачу — имея аттрактор, строить СИФ… Тогда достаточно взять аттрактор-изображение, похожее на фотографию вашей кошки и можно довольно выгодно его закодировать.

Вот здесь, собственно, начинаются проблемы. Произвольные изображения, в отличие от фракталов, не самоподобны, так что так просто эта задача не решается. Как это сделать придумал в 1992 году Арнольд Жакин, в то время — аспирант Майкла Барнсли, который считается отцом фрактального сжатия.

Самоподобие нам нужно обязательно — иначе ограниченные в своих возможностях аффинные преобразования не смогут правдоподобно приблизить изображения. А если его нет между частью и целым, то можно поискать между частью и частью. Примерно так, видимо, рассуждал и Жакин.

Упрощенная схема кодирования выглядит так:

Изображение делится на небольшие неперекрывающиеся квадратные области, называемые ранговыми блоками. По сути, разбивается на квадраты. См. картинку ниже.

Строится пул всех возможных перекрывающихся блоков в четыре раза больших ранговых — доменных блоков.

Для каждого рангового блока по очереди «примеряем» доменные блоки и ищем такое преобразование, которое делает доменный блок наиболее похожим на текущий ранговый.

Пара «преобразование-доменный блок», которая приблизилась к идеалу, ставится в соответствие ранговому блоку. В закодированное изображение сохраняются коэффициенты преобразования и координаты доменного блока. Содержимое доменного блока нам ни к чему — вы же помните, нам все равно с какой точки стартовать.

На картинке ранговый блок обозначен жёлтым, соответствующий ему доменный — красным. Также показаны этапы преобразования и результат.

Можете поиграться сами: Coder demo.

Получение оптимального преобразования — отдельная тема, однако большого труда оно не составляет. Но другой недостаток схемы виден невооруженным глазом. Можете сами подсчитать, сколько доменных блоков размером 32×32 содержит двухмегапиксельное изображение. Полный их перебор для каждого рангового блока и есть основная проблема такого вида сжатия — кодирование занимает очень много времени. Разумеется, с этим борются при помощи различных ухищрений, вроде сужения области поиска или предварительной классификации доменных блоков. С различным ущербом для качества.

Декодирование же производится просто и довольно быстро. Берем любое изображение, делим на ранговые области, последовательно заменяем их результатом применения соотв. преобразования к соотв. доменной области (что бы она ни содержала в данный момент). После нескольких итераций исходное изображение станет похоже на себя:

Пожалуй, для введения информации достаточно. Интересующиеся могут почитать соответствующие статьи в Википедии или соответствующем сообществе.

Подробное изложение можно найти в этой статье, с которой все началось в 1992 году. Большинство материалов, разумеется, на английском.

Текст 3 (выбран студентом):

Про противостояние двух главных консолей четвёртого поколения я уже упоминал в предыдущей статье, поэтому повторяться не буду. Однако, помимо плотной зарубы с Sega, Super Nintendo также оказалась тесно связана с другой знаменитой японской компанией, которая уже в следующем поколении консолей подсидела обоих и подгребла под себя весь рынок консольных игр. Конечно же, речь идёт о Sony.

Сохранившийся прототип Nintendo PlayStation. Фото Mats Lindh

Sony поучаствовала не только в дальнейшем развитии платформы, что привело сначала к созданию полумифической, не вышедшей на рынок Nintendo PlayStation, и к последующей разработке сверх-успешной Sony PlayStation, но сыграла роль и в создании самой оригинальной Super Nintendo, создав для неё звуковую систему на компонентах собственной разработки.

Легенда гласит, что в начале 1980-х годов Кен Кутараги, будущий отец PlayStation, смотрел, как его маленький ребёнок, в разных источниках сын или дочь, играл на Famicom, предыдущей игровой консоли Nintendo. В этом наблюдении Кен увидел потенциал видеоигр, которого не видела компания Sony, считая, что «игрушечный» бизнес не подходит их солидному имиджу. Кутараги в тайне от руководства самолично разработал звуковой чип под названием SPC700 и предложил его Nintendo для их следующей консоли, и хотя боссы были в ярости и чуть не уволили Кена, всё же удалось достичь соглашения, и таким образом чип разработки Sony попал в Super Nintendo. При этом производством чипа занималась сама Sony, сохранив на него права.

Все чипы звуковой системы в американской SNES

Достоверная история неизвестна, и в ней хватает тёмных пятен — и кто к кому с чем пришёл — Sony к Nintendo или наоборот, и кто на самом деле разработал чип, какого калибра фигурой был на момент его разработки Кутараги в Sony, занимался ли он сам технической частью, или уже только организационной, и мог ли он договориться от лица одной компании с другой компанией. Всё это — тайна, покрытая мраком и лежащая за языковым барьером, и наверняка всё было иначе. Неточностей в легендах тоже хватает, начиная с названия SPC700, о чём я расскажу ниже.

Первая оригинальная Sony PlayStation

Впрочем, вполне очевидно, что звуковое решение в Super Nintendo сильно отличается от всех домашних конкурентов, превосходит их технологически, реализовано очень обособленно от остальной аппаратуры консоли, а звуковая система появившейся через пять лет PlayStation концептуально очень похожа на прямое развитие решения в SNES, только с улучшенными характеристиками и без дополнительного управляющего процессора.

Реклама прошлых лет утверждала, что Super Nintendo обладает настоящим, полноценным 16-битным звуком, тогда как у её конкурента, Sega Mega Drive (Genesis), 16 бит не настоящие — дескать, видео и звук там восьмибитные. То есть король-то — восьмибитный! Разумеется, по большей части это была маркетинговая чепуха, хотя и частично построенная на сильно искажённых отзвуках реальных технических деталей. Более подробно я рассматривал этот вопрос в своей статье «Войны битов».

Реклама Super Nintendo в отечественной игровой прессе

Конечно, значительная разница в звучании определялась вовсе не количеством бит, которые ещё неизвестно где и как считать, а принципиально иным подходом к синтезу звука: собственно синтез простых тембров у Sega

Genesis и воспроизведение цифровых записей реальных инструментов у Super Nintendo. Реклама упоминала и это, но в весьма своеобразном ключе: качество звука эквивалентно CD-дискам и значительно лучше любых синтезаторов шумов, которые используются в этих ваших Сегах с Дендями.

В реальности SNES не может воспроизводить звук CD качества ни в каком виде, так как верхняя граница частоты дискретизации у неё всего 32 килогерца, и к тому же она даже не способна воспроизводить несжатое аудио без применения нетривиальных трюков. В её звуковую память может поместиться лишь 4 секунды монофонического звука в максимально возможном местном качестве, а самый ёмкий и дорогостоящий картридж в 90-х годах смог бы вместить всего одну трёхминутную стереофоническую дорожку.

И всё же, использование сэмплов реальных инструментов и применение сжатия данных позволило получить звучание, напоминающее лучшие звуковые карты для ПК того периода — Roland MT-32, Gravis Ultrasound и Sound Blaster AWE32, радикально более реалистичное, чем звучание FM-синтезаторов звуковых карт Adlib и SB16. Уровень качества местного звука также примерно сопоставим с классическими моделями компьютера Commodore Amiga, также использовавшего технологию сэмплов, с более глухим звучанием, но большей полифонией.

Архитектура

Как и в Sega Genesis, звуковая система Super Nintendo представляет собой целый отдельный микрокомпьютер с собственным процессором, памятью и синтезатором звука.

Официальная блок-схема звуковой системы

Впрочем, выделена она гораздо сильнее, чем в Sega Genesis, где шина дополнительного процессора Z80 и звуковых устройств связана с общей

системной шиной через арбитр шин, и основной процессор 68000 имеет прямой доступ к ОЗУ Z80, может перезапускать его, а также может сам управлять звуковыми устройствами — всё это было нужно для двойного назначения, не только для звука, но и для обеспечения обратной совместимости с Master System.

В SNES же звуковая система — практически чёрный ящик: полностью автономная, связанная с внешним миром, то есть остальной консолью, только посредством четырёх двунаправленных 8-битных портов ввода-вывода. Центральный процессор не может обратиться к её памяти, не может сам проиграть звук, или даже остановить или перезапустить звуковой процессор.

Во многих источниках звуковая система Super Nintendo в целом часто называется SPC700 или просто SPC. Точное происхождение этого наименования неизвестно — официальная документация лишь однажды упоминает, что в системе используется процессорное ядро серии Sony SPC700. Но называть так всю систему некорректно: SPC700 представляет собой именно ядро, систему команд процессора, которое является лишь частью всей системы, и само по себе не имеет отношения к звуку и не уникально для SNES: оно только управляет синтезатором звука.

Обзор характеристик звуковой системы в официальной документации

Система состоит из нескольких компонентов: 8-битного процессора S-SMP с архитектурой SPC700, аппаратного декодера и проигрывателя сжатых оцифрованных звуков S-DSP, общего ОЗУ объёмом 64 килобайта, а также ПЗУ объёмом 64 байта с начальным загрузчиком, так называемым IPL ROM.

При включении или перезапуске консоли процессор S-SMP начинает исполнять код загрузчика из собственного встроенного ПЗУ. Этот код обеспечивает связь с основным процессором через порты. С его помощью центральный процессор может передать звуковой системе код звукового «драйвера» в память звуковой системы и запустить его выполнение. Также в память по мере необходимости загружаются оцифрованные звуки. Как правило, это происходит при смене уровней.

Единственная точка синхронизации между системами — включение питания или сброс, коммуникация же между подсистемами чисто программная, и если что-то пойдёт не по плану, код звуковой системы зависнет, поможет только сброс — у центрального процессора нет средств возобновить его работу.

Код загруженного драйвера интерпретирует музыкальные данные, загруженные в звуковое ОЗУ и управляет S-DSP, который воспроизводит находящиеся там же сэмплы с нужной скоростью и громкостью, а также применяет характерный эффект реверберации, который является визитной карточкой игр на Super Nintendo.

Судя по всему, подобная связка из собственного стандартного ядра с архитектурой SPC500 (4-битной) или SPC700 (8-битной) и уникального для конкретной задачи DSP являлась стандартным решением для аппаратуры Sony. В частности, она использовалась в CD-проигрывателях их производства.

Чип S-APU, содержащий все компоненты звуковой системы

В оригинальной версии консоли S-SMP, S-DSP и их общее ОЗУ были представлены отдельными чипами, причём в американской версии они были смонтированы на отдельной дочерней плате. В последующих ревизиях эти компоненты были интегрированы в один чип под названием S-APU, а ещё позже вся аппаратура Super Nintendo была собрана в одном большом ASIC.

Столь обособленная архитектура звуковой системы позволила реализовать файловый формат SPC, используемый в эмуляторах Super Nintendo. Он содержит просто «снапшот» системы, содержимое звукового ОЗУ и текущее состояние регистров процессора S-SMP, которое может в любой момент сохранить эмулятор и воспроизвести отдельно стоящий проигрыватель. В один файл сохраняется одна композиция: содержимое сохранённой памяти системы настраивается таким образом, чтобы проиграть нужный трек с начала.

Такое решение не очень оптимально по расходу дискового пространства и совместимо не со всеми играми, так как некоторые из них общаются со звуковой системой в процессе игры и подгружают туда фрагменты данных музыкального трека. Тем не менее, оно до сих пор имеет большую популярность, и вот уже четверть века существует целый сайт snesmusic.org с архивами музыки, извлечённой в таком формате из множества игр для SNES.

S-SMP

Микросхема S-SMP

В официальной документации для Super Nintendo процессорное ядро звуковой системы называется Sound-CPU, но на самой микросхеме написано S-SMP. Это скорее микроконтроллер, чем процессор, так как на его кристалле помимо процессорного ядра присутствуют и периферийные устройства.

Процессор является представителем архитектуры SPC700. Насколько мне известно, это проприетарная архитектура компании Sony, применявшаяся в разнообразной аппаратуре её производства. Рискну предположить, что буквы SPC означают что-то наподобие Sony Processor Core, но это лишь мои сомнительные догадки.

Страницы каталога микроэлектронной продукции Sony за 1996 год

Так как это не контроллер общего назначения, какую-то информацию о нём можно извлечь лишь из документации на отдельные чипы Sony для различной бытовой аппаратуры и из каталогов микроэлектронной продукции компании, где упоминается наименование этой архитектуры.

У звуковой системы свой собственный генератор тактовой частоты, реализованный внутри S-DSP, а его кварцевый резонатор имеет частоту

24.576 МГц. Тактовая частота S-SMP составляет всего 1024 килогерц (1 мегагерц с копейками). Формирует её S-DSP, на нём есть соответствующий выход. Самые короткие операции (NOP и операции с регистром A) выполняются за два такта.

Опкоды SPC700 в мнемониках Sony

На первый взгляд ядро SPC700 выглядит чем-то совершенно кастомным, в основном из-за предложенной Sony системы мнемоник. Но стоит копнуть чуть-чуть глубже, и становится ясно, что это прямое развитие классического процессора MOS 6502, хотя и не совместимое на уровне бинарного кода — совпадает лишь часть опкодов.

Опкоды SPC700 в мнемониках 6502

Если записать мнемоники SPC700 в стиле 6502, архитектура становится гораздо понятнее. Она представляет собой нечто среднее между 6502 и 65816, с разнообразными полезными дополнительными операциями.

В частности, у SPC700 есть режимы 16-разрядной индексации регистровыми парами X:A и Y:A. Я бы сказал, что это расширение возможностей сделано даже лучше, чем в 65816, хотя не могу сказать ничего плохого и про него.

Регистры процессора SPC700

Вместо «Zero Page», как у 6502, тут есть аналогичная по назначению так называемая «Direct Page», которая может располагаться в нулевой или первой 256-байтной странице памяти, то есть в адресах $0000..$00FF или $0100..$01FF. Многие инструкции могут обращаться к памяти по сокращённому однобайтовому адресу внутри Direct Page, что уменьшает объём кода и работает быстрее полноценной адресации, как и у 6502.

Обе страницы ОЗУ, доступные в качестве Direct Page, частично заняты под другие нужды. По адресам $F0-FF в нулевой странице ОЗУ находятся регистры S-DSP и порты коммуникации, а в первой странице всегда располагается страница стека, как и у 6502.

Способы адресации памяти на SPC700

S-SMP подключается к ОЗУ не напрямую, а через S-DSP, который разделяет время доступа к памяти: ведь ему нужно читать из того же ОЗУ данные сэмплов для восьми каналов, не мешая работать процессору.

Физически память звуковой системы представлена двумя микросхемами PSRAM (псевдо-статическая оперативная память) объёмом 32 килобайта каждая, с организацией 32Kx8. Можно предположить, что две отдельных микросхемы памяти нужны для какого-то для разделения времени доступа. Тем не менее, их шины данных D0..D7 и адреса A0..A14 включены параллельно, раздельные только сигналы выборки чипа, для которых предусмотрены специальные линии S-DSP. Возможно на момент разработки память нужного объёма и быстродействия просто не производилась, а может быть, какое-то чередование всё же реализовано.

С объёмом памяти звуковой системы могут встречаться некоторые разночтения. Официальная документация упоминает объём 512K, подразумевая килобиты, а не килобайты — это 64 килобайта. Некоторые старые любительские исследования утверждают, что объём памяти всего 32 килобайта, или что вторые 32 килобайта зарезервированы только под буфер реверберации — всё это неверно. Дополнительную смуту вносит наличие у S-DSP незадействованной линии A15, что намекает на теоретически возможный объём звуковой памяти до 128 килобайт. Но у S-SMP дополнительных линий нет, только A0..A15.

Карта памяти звуковой системы

Из дополнительной аппаратуры на кристалле S-SMP есть три таймера и четыре порта коммуникации.

Таймеры функционально полностью одинаковы, но два из них тактируются частотой 8 кГц, а один — частотой 64 кГц. Их можно запускать и останавливать. Для каждого таймера задаётся 8-битный делитель его входной частоты, а переполнение счётчика циклически инкрементирует 4-битный регистр состояния таймера. Читать можно только эти 4-битные регистры. При чтении и переполнении регистр обнуляется, и таким образом чтением можно узнать, сколько раз переполнился счётчик за время между чтениями.

Порты устройств S-SMP

Коммуникация с основной системой происходит через четыре двунаправленных 8-битных порта. Это довольно плохо задокументированная часть системы с непонятными пограничными случаями, поэтому эмуляторы реализуют её по разному. Центральный процессор 65816 может писать 8-битные значения в четыре 8-битных регистра, а SPC700 может их читать из своих соответствующих портов. Также можно и наоборот, записывать в порты со стороны SPC700, а читать со стороны 65186.

При этом записанное одним процессором становится доступным другому, но не ему самому, и код коммуникации часто записывает только что считанный байт обратно в тот же регистр в качестве способа подтверждения получения байта. Если же оба процессора выполняют запись одновременно, возникнет конфликт с неопределённым результатом, и нужно предусматривать такую ситуацию в коде.

У SPC700 совсем нет никаких прерываний, поэтому состояние таймеров можно отслеживать только циклическим опросом. Таким же образом реализуется и коммуникация через двунаправленные порты.

IPL

Полное содержание IPL ROM

IPL — важнейшая часть, необходимая для использования звуковой системы. Это ПЗУ с кодом начального загрузчика объёмом всего 64 байта, физически расположенное внутри микросхемы S-SMP.

Код IPL ROM при включении или сбросе располагается в самых верхних адресах адресного пространства S-SMP, замещая фрагмент ОЗУ, и процессор начинает работу с его выполнения. Впрочем, есть возможность получить доступ к этим 64 байтам ОЗУ, отключив IPL программно. после старта кода.

Коммуникационные порты

Код загрузчика очищает младшие 256 байт ОЗУ звуковой системы, устанавливает стек, выдаёт в коммуникационные порты сигнатуру готовности, и ожидает получения команд от центрального процессора.

Протокол обмена, реализованный в IPL, довольно медленный, к тому же он ненадёжен и критичен к таймингам. Поэтому во время выполнения обмена со звуковой системой прерывания основного процессора должны быть запрещены, а код, передающий данные, должен быть максимально быстрым.

Официальное описание протокола

Протокол загрузки выглядит следующим образом:

IPL выполняет инициализацию и выставляет в коммуникационных портах 0..3 сигнатуру $AA BB ?? ?? (последние два байта не определены). Основной

процессор 65816, далее именуемый S-CPU, ждёт в цикле, пока в портах не появится эта сигнатура.

Когда сигнатура получена, S-CPU передаёт команду, записывая в порты 0..3 байты $CC XX YY ZZ, где последние три значения являются кодом команды и параметрами, а байт $CC является признаком команды. Записываются байты команды задом наперёд, то есть сначала адрес в порты 2-3, потом код команды в порт 1, и только потом признак $CC в порт 0.

Команд всего две: загрузить блок или запустить выполнение кода. YY и ZZ — это всегда 16-битный адрес в памяти S-SMP, а XX может быть равен 0 для запуска по этому адресу, или не 0 для загрузки блока. После этого S-CPU ждёт подтверждения команды — появления признака $CC в том же порте 0 (IPL запишет его назад).

Если передана команда пересылки блока, IPL входит в цикл приёма данных блока. Это наиболее критичная часть процесса.

S-CPU пишет первый передаваемый байт в порт 1 и значение 8-битного счётчика, который на старте передачи равен 0, в порт 0, а потом ожидает, пока значение, читаемое из порта 0, станет равно только что записанному. Первый байт передан. Далее S-CPU должен максимально быстро записать второй байт в порт 1, увеличить счётчик на 1, записать его значение в порт 0, и подождать появления такого же значения в порте 0.

Процесс продолжается циклически до передачи всех байт. Если записать в порт 0 байт меньшего значения, чем предыдущий, или выполнить передачу следующего байта чуть позже, процесс передачи прекращается. IPL снова переходит в режим ожидания признака команды. Таким образом можно передать один большой блок или много блоков разного размера по разным адресам, после чего запустить исполнение загруженного кода с нужного адреса.

Далее загруженный код реализует собственную коммуникацию с S-CPU и систему команд — запустить музыку, проиграть эффект. и так далее. У кода есть возможность снова запустить IPL и процесс передачи блоков, он может сделать это по приёму специальной команды.

Большинство игр используют стандартный протокол загрузки IPL и для подгрузки сэмплов и музыки между уровнями. Игры Super Nintendo отличается довольно медленной сменой локаций, иногда они «загружаются» удивительно долго для картриджной системы — в версии игры Batman Forever для SNES даже есть надпись «HOLD ON» при смене локаций. И скорость загрузки данных в звуковую память является одной из причин этой медлительности.

Впрочем, в идеальном случае вся звуковая память может быть загружена по стандартному протоколу за полторы секунды (один большой блок, самый оптимальный код), но в реальности этот процесс обычно медленнее. Чтобы ускорить процесс, а также сделать его более надёжным и простым, некоторые игры сразу при запуске загружают собственный код для коммуникации, и далее код звукового драйвера и все нужные данные загружаются уже с его помощью.

Также IPL позволяет реализовать трюк, позволяющий вообще не изучать архитектуру S-SMP и работать с S-DSP только с помощью основного процессора, несмотря на то, что регистры S-DSP недоступны напрямую: можно передавать с помощью IPL двухбайтовые блоки прямо в адреса регистров S-DSP для записи каждого регистра. Это далеко не самый оптимальный, но вполне рабочий вариант.

S-DSP

Микросхема S-DSP

Наконец, самая главная часть звуковой системы Super Nintendo, то, чем она издаёт звук — S-DSP. Так написано на чипе, а в документации он именуется просто DSP. Вероятно, это тот самый компонент, который разработал Кен Кутараги лично, либо инженеры Sony под его руководством.

S-DSP не является синтезатором в том смысле, что он не создаёт звук из ничего, из набора параметров, как FM или PSG-синтезаторы в Sega Genesis.

Вместо этого он воспроизводит заранее записанные фрагменты звука. Это могут быть любые реальные инструменты и синтезаторы, в том числе даже FM — в некоторых играх даже можно встретить точно такие же тембры, какие можно услышать на Genesis.

Самым главным ограничением этой системы является объём звуковой памяти: всего 64 килобайта. При обычном подходе это всего полторы секунды звучания в наилучшем качестве или восемь секунд в совершенно ужасном. Чтобы экономить драгоценную память, у S-DSP есть целый арсенал возможностей.

График из документации, иллюстрирующий ADPCM-сжатие

Прежде всего, это аппаратное сжатие сэмплов с потерями. Они хранятся в местной версии формата ADPCM, называемого в официальной документации BRR, Bit Rate Reduction. Сжатие работает пакетами по 16 монофонических 16-битных отсчётов, кодируя их в блоки размером 9 байт. Коэффициент сжатия получается фиксированным, 3.56:1. Кодировка сэмплов производилась программными средствами разработки Sony, так как эта компания долгое время, до широкого внедрения MP3, была пропонентом вариаций формата APDCM.

Максимальная частота дискретизации сэмплов, поддерживаемая S-DSP — 32 килогерца. В реальности игры использовали меньшие частоты, отличающиеся для каждого сэмпла в зависимости от характера его содержания. Это также делалось для оптимизации расхода памяти.

Устройство одного канала S-DSP

Чтобы не хранить сэмплы каждой нужной для мелодии ноты и каждого инструмента, S-DSP позволяет получать их из одного и того же сэмпла, меняя скорость воспроизведения: чем быстрее воспроизводится звук, тем выше он

звучит, а чем медленнее — тем ниже. Градаций возможных скоростей очень много, что позволяет точно попадать в довольно широкий диапазон нот.

Конечно, такой способ изменения высоты меняет гармонический состав звука и длительность его звучания, что придаёт звукам очень характерную окраску. Профессиональные синтезаторы с аналогичным принципом работы используют несколько сэмплов для каждого инструмента, каждый на маленький диапазон нот, что делает звучание значительно более реалистичным, но и требует значительно больше памяти. На SNES подобный подход встречается редко.

S-DSP умеет воспроизводить звук только вперёд, а также зацикливать часть сэмпла, но только по границам пакетов, что вносит определённые сложности в подготовку зацикленных сэмплов: позиции обеих точек цикла должны быть кратны 16 отсчётам.

ADSR-огибающая S-DSP

Также S-DSP поддерживает стандартную амплитудную огибающую формата ADSR: можно выбрать 16 скоростей нарастания до максимума, затухания до указанного уровня и полного затухания сэмпла, а также скорость затухания при досрочной остановке. Это позволяет делать более интересно звучащие инструменты с коротким циклом, то есть занимающие мало памяти.

С учётом ограниченной памяти возможности звуковой системы Super Nintendo хорошо подходят для имитации реалистичных аранжировок, оркестровок, музыки с отрывистым звучанием. Не очень хорошо работают протяжённые во времени тембры, память не позволяет хранить длинные звуки, а S-DSP не позволяет использовать двунаправленный цикл, лучше маскирующий точку зацикливания.

Again https://www.youtube.com/watch?v=EvA55UUGamkТакже из-за применения сжатия общее звучание часто довольно глухое. Не очень хорошо звучат синтезаторные звуки, рок-музыка и гитары из-за смены гармоник при изменении высоты ноты звучат весьма своеобразно. Впрочем, и техно, и

эмбиент, тоже могут звучать очень хорошо. Ведь как и всегда, прежде всего всё зависит не от техники, а от таланта композиторов.

Карта регистров S-DSP

S-DSP управляется примерно сотней регистров, организованных довольно логичным и удобным образом. В памяти S-SMP для доступа к ним отведены всего два адреса: в один записывается номер регистра S-DSP, в другой — значение для выбранного регистра.

Каждому из восьми голосов отведено по 9 регистров: громкость левая и правая, 14-битная скорость воспроизведения (высота звука), номер сэмпла в таблице, 4-битные параметры ADSR-огибающей.

Также есть общие настройки S-DSP: глобальная громкость (левый и правый каналы), громкость эффекта реверберации, запуск и остановка каналов, настройки реверберации (расположение буфера, параметры фильтра), указатель на список сэмплов, и некоторые другие

Ещё у S-DSP есть пара интересных, но преступно редко используемых возможностей: генератор белого шума и эффекты модуляции одного канала другим, что создаёт некоторое подобие FM-синтеза.

Генератор шума один, общий для всех каналов. Он имеет 32 варианта «высоты» шума (аналогично PSG). Шум может быть включён в любом из восьми каналов вместо сэмпла-источника, и к нему также можно применять модуляцию. Его использование можно слышать в играх Capcom, Squaresoft и других компаний. В F-Zero это шум двигателя, в Super Mario RPG — звук водопада, а в Donkey Kong Country — шипение змей.

Работу эффекта модуляции можно услышать ушами в играх Secret of Mana и Waterworld. Модуляция использует пару каналов, и её можно включить для любого из для семи каналов. Когда она включена, предыдущий канал становится модулятором. При этом можно выставить его громкость в 0, чтобы не слышать модулирующий звук, а можно оставить слышимым. В зависимости от громкости канала-модулятора меняется скорость проигрывания другого. Чем выше громкость, тем сильнее влияние. Таким образом, сэмпл синусоиды в одном канале позволяет создать эффект вибрато для любого произвольного сэмпла в другом канала, а скорость проигрывания сэмпла в канале с синусоидой будет определять скорость вибрато.

Наконец, S-DSP реализует весьма узнаваемый, характерный эффект звучания игр на Super Nintendo — реверберацию. Её можно слышать во множестве игр с разными настройками. Фактически это скорее не реверберация, а подвид эхо, «дилея» — эффект задержки с циклической записью обратно в монофонический буфер с применением 8-точечного FIR-фильтра. Громкость сигнала, поступающего из буфера на выход, регулируется отдельно.

Под буфер реверберации задействуется часть общего звукового ОЗУ, чем дольше задержка, тем больше нужно памяти под буфер. Длительность задержки составляет от 16 до 240 миллисекунд, при этом размер буфера может достигать 30 килобайт. Буфер стереофонический, в нём хранятся 16-битные несжатые отсчёты. При работающем эффекте буфер постоянно перезаписывается, поэтому необходимо следить, чтобы он не наложился на область хранения кода и данных.

Также возможно сделать трюк и применить фильтр реверберации к основному выходу звука. Для этого нужно установить сверх-короткий буфер длиной в один отсчёт (фильтр имеет свой буфер вне ОЗУ), убрать громкость основного выхода, установить полную громкость выхода реверберации. Этот же трюк позволяет воспроизводить несжатое аудио, например, синтезируемое на S-SMP или получаемое из картриджа сверх-большого объёма: 16-битные стерео отсчёты можно записывать прямо в буфер реверберации, откуда их подхватит S-DSP и отправит на выход.

Комбинация всех доступных S-DSP эффектов и возможностей сильно обогащает звучание коротких сэмплов в играх, создавая характерное узнаваемое звучание Super Nintendo.

Звуковая система PlayStation хотя и отличается в деталях реализации, но концептуально представляет собой развитый количественно (число каналов, объём памяти) вариант S-DSP. Главным её отличием является отсутствие аналога S-SMP, его роль там играет центральный процессор.

Драйверы

Как и на Sega Genesis, и на других консолях, сама по себе звуковая система не умеет воспроизводить музыку, и просто ничего не знает о такой высокоуровневой концепции — она оперирует отдельными звуками. Превращает набор сэмплов и поток музыкальных данных в музыку так называемый «драйвер», загружаемая в звуковую память программа, выполняемая S-SMP и управляющая S-DSP.

На Sega Genesis драйверы сильно отличались по возможностям и качеству звука: драйвер мог поддерживать или не поддерживать часть возможностей железа, в том числе приличную часть каналов синтезатора.

На Super Nintendo нет такого разнообразия возможностей, и все драйверы поддерживают основную их часть, отличаясь в небольших деталях и организации логики высокого уровня, особенно в части менеджмента каналов. На низком уровне главные отличия заключаются в наличии поддержки генератора шума и модуляции — редко используемых возможностей железа. Качество же звучания музыки больше зависит от качества подготовки и выбора сэмплов, чем от возможностей звукового драйвера.

И тем не менее, разных звуковых драйверов на Super Nintendo существуют сотни. Энтузиасты составили огромный их список, прочитать который можно здесь.

Одним из самых главных и часто используемых драйверов является так называемый N-SPC. Название это, вероятно, было изобретено энтузиастами. Драйвер был частью пакета разработки самой Nintendo, включавшей ПО для создания звуков и музыки на компьютерах Sony NEWS (рабочая Unix-станция на процессоре 68030).

По обрывочной информации, в пакет входили программы Tako Sample для записи и Ika Listen для прослушивания получившихся сэмплов на железе. Музыкальный текст готовился в секвенсоре «Kankichi-kun», позволяющем записывать партии с MIDI-входа и оптимизировать их для использования в играх, или в формате текстового MML-скрипта. Более точных сведений или самого ПО энтузиастами пока обнаружено не было.

Ответственен за создание драйвера предположительно Юкио Канэока, композитор и разработчик компании Nintendo, сочинивший немало музыки для игр на Famicom, а последний раз его имя упоминается в игре F-Zero 1990 года для SNES, в которой он вероятно делал именно код, но не сочинял музыку.

Изначально это был не готовый драйвер, а просто пример кода, показывающий воспроизведение только музыки. Так как других примеров не было, а задача сложная — освоить целый новый процессор и DSP — многие разработчики, и японские, и западные, брали его за основу и дорабатывали для нужд конкретных игр, добавляя поддержку возможностей и звуковых эффектов.

Девкит от Intelligent Systems, включающий IS-SOUND

Также известно, что вариант драйвера N-SPC, доработанный Кеничи Нишимаки, применялся в девките IS-SOUND разработки Intelligent System, дочерней компании Nintendo, которая разработала специализированные рабочие станции для создания музыки, графики и кода для SNES, и потом в его обновлённой интегрированной версии «Super NES Emulator» (название подразумевает аппаратный эмулятор).

Не у всех разработчиков был доступ к полному комплекту ПО, а только исходник драйвера N-SPC и одиночные сэмплы. Тогда они писали собственные конвертеры из удобных им форматов, или саму музыку прямо в ассемблерном исходнике, в виде множества блоков типа DB, а также дорабатывали драйверы под свои нужды.

Всёэто привело к появлению доброй сотни подвидов одного только этого драйвера. Разбор получившегося хитроумного генеалогического древа — задачка, достойная отдельной статьи.

Трекер Octamed на Amiga

Отдельный интересный случай — опыт Альберто Гонзалеса, который имел доступ к IS-SOUND и Kankichi-kun, но сочинял музыку в более удобном для него трекере Octamed на компьютере Amiga, после чего воспроизводил музыкальные партии из него через MIDI-выход компьютера и таким образом переносил их в секвенсор.

Аппаратная часть системы SLICK/Audio

Также довольно широко использовалась звуковая система SLICK/Audio разработки Bitmasters. Она состояла из программного обеспечения и карты расширения для ПК, кабеля и специального картриджа. Концептуально работа с ней была похожа на GEMS для Sega Genesis: подготавливаемую с помощью MIDI-партий на ПК музыку можно было сразу прослушать на реальной консоли.

Другие разработчики и компании тоже делали свои трекеры, конвертеры, а также полностью оригинальные драйверы, которых существует также довольно много, около сотни. Однако, различить их все на слух гораздо сложнее, чем в случае с Sega Genesis, поэтому отдельного разбора примеров звучания я не привожу.

Отличия драйверов могут заключаться в архитектуре: одни играют музыку самостоятельно, а другие только транслируют команды для передачи S-DSP, которые подаёт центральный процессор. Данные музыки и сэмплы могут загружаться однократно, а могут динамически подгружаться по мере надобности. Некоторые драйверы умеют передавать поток звуковых данных из ПЗУ картриджа — так сделана впечатляющая вокальная партия в начальной заставке игры Tales of Phantasia.

Также отличия касаются реализации звуковых эффектов. Многие драйверы не умеют динамически разделять каналы между звуковыми эффектами и музыкой, выделяя, например, шесть каналов на музыку и два на эффекты. Другие способны прерывать звучание каналов музыки на время проигрывания эффекта. Некоторые драйверы реализуют эффекты чисто сэмплами, а другие используют секвенции и несколько простых сэмплов, подобно звуковым эффектам на 8-битных компьютерах (например, игры Capcom).

Ещё одно заметное отличие касается применения реверберации.Некоторые игры накладывают её на весь звук, другие не используют вовсе, третьи меняют динамически в процессе игры — как в Donkey Kong Country.

Наши дни

Удивительно, но любительская разработка для Super Nintendo взяла старт очень рано, ещё во времена активной коммерческой жизни платформы в

начале 90-х годов, даже до появления первых программных эмуляторов. Случилось это благодаря появлению «копировщиков», устройств, позволяющих сохранять образы картриджей на дискеты и загружать их.

С появлением эмуляторов, впрочем, энтузиазм к платформе упал, так как к тому времени она уже перешла в категорию ретро, и к тому же оказалась весьма сложной в освоении. Несмотря на устойчивый интерес энтузиастов к ретро-деву, для Super Nintendo до сих пор появляется очень мало любительских проектов, в частности, из-за отсутствия удобных и надёжных средств разработки. У Sega Genesis с этим гораздо лучше.

Тем не менее, за годы существования homebrew-сцены были созданы разнообразные решения в том числе и для озвучивания игр или создания отдельно прослушиваемой аутентичной музыки в стиле Super Nintendo, и я коротко пройдусь по основным из них.

Начиналось всё с конвертеров из трекерных форматов: XMSNES (2006), IT2SPC и его наследник SNESMOD (2009). Эти конвертеры принимают созданные по определённым правилам 8-канальные трекерные модули, с рядом ограничений, и конвертируют их в SPC-файл. Не самое подходящее для игр решение, так как каждый музыкальный трек использует свой набор сэмплов и отсутствует (развитая) поддержка звуковых эффектов.

Когда я занялся разработкой для SNES в 2011 году, на первое время я тоже разработал аналогичное решение, с конвертором XM-модулей, но с собственным драйвером, поддержкой мульти-трековых композиций — несколько мелодий в одном файле с общим набором сэмплов, и поддержкой секвенций для звуковых эффектов. Иначе говоря, эффекты тоже представляют собой мелодии, только короткие и одноканальные: это позволяет создавать разнообразные сложные звуки, используя всего несколько коротких сэмплов.

Интерфейс SNES GSS последней версии

Позже, в 2014 году, на основе элементов своего предыдущего драйвера я разработал более продвинутую систему SNES GSS (Game Sound System) для игр, которые делал для BubbleZap. Она имеет свой кросс-трекер, конвертер сэмплов, и поддержку всего, что необходимо для игр. Из недостатков — минималистичный пользовательский интерфейс, в нём есть только то, что действительно необходимо для решения задачи.

Интерфейс snestracker

Была попытка создания кросс-трекера под названием snestracker, велась активная PR-компания с платными бета-версиями, футболками и планами на кампанию на Kickstarter. К сожалению, этот проект развивается крайне медленно. Он был начат десять лет назад, в 2015 году, а последняя демо-версия вышла пару лет назад и показывала не самый убедительный результат — частично функциональный редактор с примитивным полутекстовым интерфейсом.

Интерфейс Furnace и трек для SNES

Конечно, самый развитый на данный момент кросс-трекер с поддержкой SNES — многоплатформенный трекер-комбайн, Furnace. К сожалению, он до сих пор страдает значительной проблемой: хотя в нём поддерживается невероятное количество звуковых чипов и платформ, для большинства из них не реализован экспорт в применимые на реальных устройствах форматы. В том числе не предусмотрен экспорт и для SNES. Впрочем, существует попытка решения этой проблемы в лице конвертера furSPC, но полноценная поддержка возможностей трекера в нём пока не реализована.

Помимо трекеров, есть и отдельные звуковые драйверы. Так, в 2017 году был создан qSPC, компилятор MML-скриптов для SNES, ориентированный на создание просто музыки. А в 2023 году появился аналогичный проект с компилятором MML — Terrific Audio Driver, но уже предусматривающий применение в играх. Здесь также применяется идея с общим набором

сэмплов. Для звуковых эффектов доступно только два канала, предусмотрена поддержка 8-канальной музыки с заглушением каналов на время воспроизведения звуковых эффектов.

Редактор EbMusEd

Есть редакторы, предназначенные для замены музыки в отдельных играх, ориентированные на конкретные коммерческие драйверы прошлых лет. Например, EbMusEd — редактор музыки для игры EarthBound, использующей один из подвидов драйвера N-SPC. Но они не подходят для новых разработок по причине существования авторских прав: Nintendo весьма ревностно относится к своей интеллектуальной собственности, даже настолько древней.

Интерфейс VSTi-плагина C700

Также существуют и другие способы создания музыки для реальной SNES. Например, VSTi-плагин C700 для современных цифровых студий. Он эмулирует звуковую систему и загруженные в её память BRR-сэмплы, позволяет управлять ей при помощи MIDI-дорожки, а также захватывать управляющую дорожку, конвертировать в нужный вид, и экспортировать SPC-файл, который далее можно проиграть на самой консоли.

Помимо музыкальных редакторов, несколько слов стоит уделить проекту MSU-1. Это виртуальный «чип расширения», призванный дать SNES возможности, аналогичные тем, которые изначально были заложены в проекте Nintendo PlayStation: поддержку носителя большого объёма (до 4 гигабайт), воспроизведение звука CD-качества и видеороликов. Изначально этот проект был реализован в эмуляторах, и для него было адаптировано несколько игр. В основном они заменяют синтезируемый звуковой системой саундтрек на полноценные аудиотреки высокого качества, но есть также игры с видеовставками. Позже поддержка игр для MSU-1 была реализована и на

Flash-картридже SD2SNES, и таким образом их можно запускать даже на реальном железе.

Заключение

Тема звуковых возможностей двух основных 16-битных платформ раскрыта. Но цикл на этом не обязательно завершается. Ведь есть и другие платформы с другим количеством бит, и с по своему интересными звуковыми возможностями. Например, всеми любимый ПК и ранний период становления концепции «звуковой карты». Или отечественные ПК и их своеобразный подход к реализации звука. А значит, есть поводы для продолжения цикла!

Текст 4 (выбран студентом):

В этой статье я покажу как можно самому, бесплатно и без смс, нарисовать черную дыру при помощи OpenGL, в полном соответствии с ОТО.

Для этого, мы сначала выведем уравнения движения лучей света, напишем интегратор Рунге-Кутты на GLSL, и наконец, объединив одно с другим, получим фрагментный шейдер, который вычисляет путь лучей, отправленных из камеры назад во времени.

Подготовка: метрика, связность и геодезические

Гравитационное линзирование

Гравитационное линзирование

Как известно, свет распространяется по прямой линии. Это означает, что вектор скорости не меняется при параллельном переносе вдоль самого себя.

Однако «прямая» — понятие относительное. Так, уравнение прямой в полярных координатах выглядит уже совсем не линейным. В случае же, когда и само пространство не плоское (к примеру, на поверхности тора), выбрать систему координат, в которой «прямая» определяется линейным уравнением, попросту невозможно.

"Прямая" на торе

"Прямая" на торе

Как тогда можно вообще считать некую кривую «прямой»? Самое логичное — взять вектор в любой её точке, направленный по касательной, и начать его переносить вдоль этой кривой. Если он при любом таком переносе останется направленным по касательной — то кривая называется геодезической, и это самое близкое что есть к понятию прямой на произвольном многообразии.

Выводить формулы для геодезических на четырехмерном многообразии — довольно трудоемкое занятие. К счастью, это как раз тот случай, когда труд можно переложить на машину. Воспользуемся библиотекой символьной алгебры sympy.

```
from sympy import *
from sympy.diffgeom import *
import numpy as np
```

Модуль sympy.diffgeom довольно нетривиален в использовании и, я бы даже сказал, уродлив, но он неплохо справляется с задачей вывода всяких уравнений на многообразиях, описанных атласом. Наш атлас будет иметь всего одну карту, но с двумя разными системами координат - сферической (т.к. именно в ней удобно работать с метрикой Шварцшильда), и декартовой - для рисования на экране.

Объявим символы для координат:

```
t_, x_, y_, z_ = symbols("t x y z")
rho_, theta_, phi_ = symbols("\\rho \\theta \\phi")
```

Я использую постфикс _ для терминальных символов, чтобы их было сразу видно в коде.

Составим уравнения перехода из сферических координат в декартовы:

```
x = rho_ * sin(theta_) * sin(phi_)
y = rho_ * sin(theta_) * cos(phi_)
```

```
z = rho_ * cos(theta_)

print(latex(x_) + " = " + latex(x))

print(latex(y_) + " = " + latex(y))

print(latex(z_) + " = " + latex(z))
```

```
\begin{array}{l} x = \rho \sin{\left(\phi \right)} \sin{\left(\theta \right)} \\ y = \rho \sin{\left(\theta \right)} \cos{\left(\phi \right)} \\ z = \rho \cos{\left(\theta \right)} \\ \end{array}
```

Создадим 4х-мерное многообразие spacetime, добавим в его атлас карту patch, зададим на ней две наши координатные системы, и добавим возможность переходить из сферической системы в декартову:

```
spacetime = Manifold("spacetime", 4)

patch = Patch("patch", spacetime)


relation_dict = {
    ("spherical", "cart"): [(x_, y_, z_, t_), (x, y, z, t_)]
}


coord_sh = CoordSystem("spherical", patch, (t_, rho_, theta_, phi_), relation_dict)

coord_cart = CoordSystem("cart", patch, (x_, y_, z_, t_), relation_dict)
```

Одно из неудобств sympy.diffgeom заключается в том, что уже заданные символы координат нельзя использовать для работы с многообразием. Нужно взять оттуда новые:

```
c_t, c_rho, c_theta, c_phi = coord_sh.base_scalars()

print(",".join(map(latex, (c_t, c_rho, c_theta, c_phi))))
```

```
\mathbf{t},\mathbf{\rho},\mathbf{\theta},\mathbf{\phi}
```

Как видно, это те же самые символы, которые мы задавали в начале, но пригодные для использования в sympy.giffgeom.

Зададим на нашем многообразии метрику Шварцшильда:

```
Rs = symbols("r_s")


d_t, d_rho, d_theta, d_phi = coord_sh.base_oneforms()


TP = TensorProduct
metric = (
    + (1 - Rs/c_rho) * TP(d_t, d_t)
    - 1 / (1 - Rs / c_rho) * TP(d_rho, d_rho)
    - (c_rho**2) * TP(d_theta, d_theta)
    - (c_rho**2) * (sin(c_theta)**2) * TP(d_phi, d_phi))


print(latex(metric))
```

```
\left(- \frac{r_{s}}{\mathbf{\rho}} + 1\right) \operatorname{d}t \otimes \operatorname{d}t - \sin^{2}{\left(\mathbf{\theta} \right)} \mathbf{\rho}^{2} \operatorname{d}\phi \otimes \operatorname{d}\phi - \mathbf{\rho}^{2} \operatorname{d}\theta \otimes \operatorname{d}\theta - \frac{\operatorname{d}\rho \otimes \operatorname{d}\rho}{- \frac{r_{s}}{\mathbf{\rho}} + 1}
```

Необходимость выписывать метрику в таком виде - это еще одно неудобство simpy.diffgeom. Казалось бы, дайте возможность скормить туда компоненты тензора g_{ij}, но нет. Только дифференциальные формы, только хардкор!

Как бы то ни было, вооружившись метрикой в таком виде, можно рассчитать коэффициенты метрически-совместимой связности (также известные как символы Кристоффеля второго рода). Метрически-совместимая связность — это такой способ движения из касательного пространства в одной точке к касательному пространству в другой точке, при котором метрический тензор инвариантен. То есть его ковариантная производная равняется нулю (но компоненты тензора в криволинейной системе координат, разумеется, могут меняться).

Модуль simpy.diffgeom услужливо предоставляет функцию для вычисления символов Кристоффеля metric_to_Christoffel_2nd:

```
christoffel = simplify(metric_to_Christoffel_2nd(metric))
```

```
print("\\begin{cases}")
for (i,j,k) in np.ndindex(*np.shape(christoffel)):
    if not christoffel[i,j,k].is_zero:
        print(f"\\Gamma_{{{i}{j}{k}}} =" + latex(christoffel[i,j,k]) + " \\\\")
print("\\end{cases}")
```

```
\begin{cases}\Gamma_{001} =\frac{r_{s}}{2 \left(- r_{s} + \mathbf{\rho}\right) \mathbf{\rho}} \\\Gamma_{010} =\frac{r_{s}}{2 \left(- r_{s} + \mathbf{\rho}\right) \mathbf{\rho}} \\\Gamma_{100} =\frac{r_{s} \left(- r_{s} + \mathbf{\rho}\right)}{2 \mathbf{\rho}^{3}} \\\Gamma_{111} =\frac{r_{s}}{2 \left(r_{s} - \mathbf{\rho}\right) \mathbf{\rho}} \\\Gamma_{122} =r_{s} - \mathbf{\rho} \\\Gamma_{133} =\left(r_{s} - \mathbf{\rho}\right) \sin^{2}{\left(\mathbf{\theta} \right)} \\\Gamma_{212} =\frac{1}{\mathbf{\rho}} \\\Gamma_{221} =\frac{1}{\mathbf{\rho}} \\\Gamma_{233} =- \frac{\sin{\left(2 \mathbf{\theta} \right)}}{2} \\\Gamma_{313} =\frac{1}{\mathbf{\rho}} \\\Gamma_{323} =\frac{1}{\tan{\left(\mathbf{\theta} \right)}} \\\Gamma_{331} =\frac{1}{\mathbf{\rho}} \\\Gamma_{332} =\frac{1}{\tan{\left(\mathbf{\theta} \right)}}\end{cases}
```

А вот вывод уравнения геодезической придется написать самим. Он, к счастью, относительно несложный.

Пусть кривая задана параметрически, как x^\mu(t). Тогда касательный к ней вектор имеет координаты \dot{x}^\mu(t) = \frac{d}{dt}x^\mu(t). По определению геодезической, ковариантная производная касательного вектора вдоль самого себя равняется нулю:

```
\nabla_{\dot{x}} \dot{x} = 0
```

Раскроем ее через символы Кристоффеля:

$$\nabla_{\dot{x}} \dot{x}=\left(\frac{\partial}{\partial{x^\mu}} \dot{x}^\nu + \Gamma^{\nu}_{\kappa\mu}\dot{x}^\kappa \right) \dot{x}^\mu=\left(\frac{\partial}{\partial{x^\mu}} \frac{dx^\nu}{dt} \right) \frac{dx^\mu}{dt}+\Gamma^{\nu}_{\kappa\mu} \frac{dx^\kappa}{dt} \frac{dx^\mu}{dt}$$

В первом слагаемом поменяем порядок множителей и воспользуемся цепным правилом:

$$\left(\frac{\partial}{\partial{x^\mu}} \frac{dx^\nu}{dt} \right) \frac{dx^\mu}{dt}=\frac{dx^\mu}{dt}\frac{\partial}{\partial{x^\mu}} \left(\frac{dx^\nu}{dt} \right)=\frac{d}{dt}\left(\frac{dx^\nu}{dt} \right)=\frac{d^2 x^\nu}{dt^2}$$

Тогда всё уравнение сводится к

$$\frac{d^2 x^\nu}{dt^2} + \Gamma^{\nu}_{\kappa\mu} \frac{dx^\kappa}{dt} \frac{dx^\mu}{dt}= 0$$

Или, проще говоря

$$\boxed{\ddot{x}^\nu =-\Gamma^{\nu}_{\kappa\mu} \dot{x}^\kappa \dot{x}^\mu}$$

На питоне:

```
def geo_second_derivatives(christoffel, derivatives):
    '''
    Parameters:
        christoffel: tensor of rank 3 - analytical expression of christoffel symbol in given coordinates
        derivatives: iterable consisting of sympy symbols corresponding to each first derivative of a coordinate variable
    '''
    d2u = np.zeros(len(derivatives), dtype='object')
    for n,k,m in np.ndindex(*np.shape(christoffel)):
```

```
        d2u[n] -= christoffel[n,k,m] * derivatives[k] * derivatives[m]
    return d2u
```

d2_t, d2_rho, d2_theta, d2_phi = geo_second_derivatives(christoffel, symbols("t' \\rho' \\theta' \\phi'"))

print("\\begin{array}{l}")

print(f"{{{latex(c_t)}}}" = {latex(d2_t)} \\\\")

print(f"{{{latex(c_rho)}}}" = {latex(d2_rho)} \\\\")

print(f"{{{latex(c_theta)}}}" = {latex(d2_theta)} \\\\")

print(f"{{{latex(c_phi)}}}" = {latex(d2_phi)} \\\\")

print("\\end{array}")

\begin{array}{l} {\mathbf{t}}" = - \frac{\rho' r_{s} t'}{\left(- r_{s} + \mathbf{\rho}\right) \mathbf{\rho}} \\ {\mathbf{\rho}}" = - \phi'^{2} \left(r_{s} - \mathbf{\rho}\right) \sin^{2}{\left(\mathbf{\theta} \right)} - \frac{\rho'^{2} r_{s}}{2 \left(r_{s} - \mathbf{\rho}\right) \mathbf{\rho}} - \theta'^{2} \left(r_{s} - \mathbf{\rho}\right) - \frac{r_{s} t'^{2} \left(- r_{s} + \mathbf{\rho}\right)}{2 \mathbf{\rho}^{3}} \\ {\mathbf{\theta}}" = \frac{\phi'^{2} \sin{\left(2 \mathbf{\theta} \right)}}{2} - \frac{2 \rho' \theta'}{\mathbf{\rho}} \\ {\mathbf{\phi}}" = - \frac{2 \phi' \rho'}{\mathbf{\rho}} - \frac{2 \phi' \theta'}{\tan{\left(\mathbf{\theta} \right)}} \\ \end{array}

Так мы получили систему дифференциальных уравнений, описывающих координаты геодезических линий.

Трассировка лучей

Раздвоение изображения полюса сферы

Раздвоение изображения полюса сферы

Итак, лучи света (а также и любые другие объекты в гравитационном поле) следуют по "прямым" линиям, заданным дифференциальным уравнением второго порядка. Рассчитать путь одного такого лучика можно легко и непринужденно на том же питоне, воспользовавшись scipy.integrate.

Но что если мы хотим увидеть черную дыру? Если отправить по лучику света из одной точки через каждый пиксель экрана, то они обернутся вокруг черной дыры, и упадут на то что расположено вокруг нее. А если они при этом будут обращены во времени, то получится что мы как бы приняли лучи, испущенные объектами в прошлом!

Но отправить миллион параллельных лучей — небыстрое занятие. Если только мы не можем это сделать параллельно на какой-нибудь крутой вычислительной машине с кучей ядер. Именно для таких задач существуют GPU.

Для исполнения одной и той же задачи на каждый пиксель окна есть специальный тип программ для GPU — фрагментные шейдеры. Для написания своего шейдера возьмем OpenGL, и его язык GLSL.

План предельно простой — разместить цветную сферическую оболочку вокруг черной дыры, разместить внутри нее камеру, и отправить из камеры лучи, которые либо упадут на эту оболочку, окрасив пиксель экрана в соответствующий цвет, либо потеряются где-то на горизонте событий.

Уравнение геодезической в GLSL...

Перепечатывать уравнения, которые были получены в sympy, в другой язык, конечно, мало кому хочется, но к счастью, там есть функция ccode, а в любом текстовом редакторе есть автозамена наших latex-фредли переменных на текстовые. После небольшого причесывания руками, уравнение геодезической p'' = f(p, p') на GLSL выглядит так:

```
vec4 d2p(vec4 p, vec4 dp)
{
    float t = p[0];
    float rho = p[1];
    float theta = p[2];
    float phi = p[3];
```

```
    float D_t = dp[0];
    float D_rho = dp[1];
    float D_theta = dp[2];
    float D_phi = dp[3];

    // Geodesics equation:
    float D2_t = - (D_rho * rs * D_t) / (rho * (rho - rs));
    float D2_rho = (rho - rs) * (
        + pow(D_phi * sin(theta), 2)
        + pow(D_theta, 2)
        - (rs * pow(D_t, 2)) / (2 * pow(rho, 3))
    )
    + (rs * pow(D_rho, 2)) / (2 * rho * (rho - rs));
    float D2_theta = pow(D_phi, 2) * sin(theta) * cos(theta) - (2 * D_rho * D_theta)
/ rho;
    float D2_phi = -(2 * D_phi * D_theta) / tan(theta) - (2 * D_rho * D_phi) / rho;

    vec4 d2p = {D2_t, D2_rho, D2_theta, D2_phi};
    return d2p;
}
```

Эта функция принимает на вход координаты точки луча и 4-вектор скорости движения вдоль него. Возвращаемое значение — вторая производная координат при движении вдоль луча.

Здесь сразу можно заметить проблему — видите деление на tan(\theta)? Когда этот угол равняется нулю, значение второй производной будет неопределено. Также, когда угол \theta околонулевой — точность интегрирования просядет. К счастью, это не такая большая проблема. Мы ее решим при помощи переменного шага интегрирования. Альтернативно, можно было бы вспомнить что для невращающейся черной дыры движение луча никогда не покидает одну плоскость, и оптимизировать вычисления таким образом. Но мне лень это я оставлю в качестве упражнения читателю.

Итак, теперь мы можем вычислить вторую производную любой геодезической, проходящей через точку p с 4-скоростью dp. Эта вторая производная в свою очередь показывает как изменяется 4-скорость при проходе через эту точку, а 4-скорость определяет в какую следующую точку мы попадем. Классическая задача для численного интегрирования.

... и его численное решение

В поиске хрупкого баланса между точностью и производительностью, я перепробовал несколько разных методов интегрирования и остановился на адаптивном методе Рунге-Кутты 3-4 порядков.

Суть адаптивных методов Рунге-Кутты заключается в том, что параллельно работают два метода разных порядков (в данном случае 3-го и 4-го). Разница между полученными значениями позволяет оценить величину погрешности, и скорректировать шаг интегрирования: если она слишком большая - уменьшить, если достаточно маленькая - увеличить.

Вдвойне удобно то что промежуточные результаты метода одного порядка могут быть переиспользованы с другими коэффициентами внутри метода другого порядка, таким образом увеличивая производительность. Подбор этих коэффициентов - целая наука и на эту тему написана не одна сотня научных статей.

Я эти сотни статей читать, конечно же, не стал. По рабоче-крестьянски я взял классический RK4 как основной метод, и эмпирически подогнал метод третьего порядка чтобы по-максимуму переиспользовать уже посчитанное.

О методе RK4

Но погодите, ведь RK4 решает уравнения 1-го порядка, а у нас-то - второго! К счастью, это не проблема. Любое уравнение вида

$$p'' = f(\tau, p, p')$$

можно свести к уравнению первого порядка, увеличив размерность и рассматривая {} p' как отдельную независимую переменную:

y =\begin{bmatrix}p \\ p'\end{bmatrix}

тогда

y' =\begin{bmatrix}p \\ p'\end{bmatrix}'=\begin{bmatrix}p' \\ p''\end{bmatrix}=\begin{bmatrix}p' \\ f(\tau,p,p')\end{bmatrix}=g(\tau,y)

Это уравнение — уже первого порядка, а значит мы умеем его решать.

Обернем функцию d2p для использования в таком виде:

```
void rk4_diff(vec4 y[2], out vec4 dy[2])
{
   dy[0] = y[1];
   dy[1] = d2p(y[0], y[1]);
}
```

Эта обертка на вход принимает 8-мерный вектор, включающий в себя как положение точки, так и ее скорость, а на выходе отдает его производную: скорость точки как производную от положения, и вычисленное функцией d2p «ускорение».

Итоговый код интегратора получился довольно-таки страшным:

rk34_step

Координаты

Черная дыра, заключенная в сферу

Черная дыра, заключенная в сферу

Итак, мы научились (в сферической системе координат) брать начальное положение и 4-скорость и кормить их в интегратор rk34_step. Эта функция

делает небольшой шажок по геодезической, и считает какой должен быть размер у следующего шага чтобы погрешность оставалась в районе заданной.

Теперь у нас есть все инструменты чтобы отправить лучи света куда угодно! Но пока нет инструментов, чтобы понять куда же их отправлять.

А для этого, так же как и раньше, нам понадобится две системы координат: сферическая для вычислений, и декартова — для работы с камерой. И возможность переходить между ними.

Преобразование координат точек

Мы уже умеем преобразовывать координаты точек из сферической системы координат в декартову. Просто перенесем это преобразование на GLSL:

```
vec4 sh_to_cart(vec4 sh)
{
    float t = sh[0];
    float r = sh[1];
    float th = sh[2];
    float phi = sh[3];

    float x = r*sin(th)*sin(phi);
    float y = r*sin(th)*cos(phi);
    float z = r*cos(th);

    vec4 ret = {x, y, z, t};
    return ret;
}
```

У декартовых координат, в отличие от сферических, время выбрано последней - для того чтобы к x,y,z можно было обращаться через точку как

обычным компонентам GLSL-ного 4-вектора: point.x, point.y, point.z. К t можно будет обращаться через point.w.

Для того, чтобы определить куда именно отправлять наш луч, пригодится обратное преобразование: из декартовых координат (пикселей экрана) в сферические.

Такое обратное преобразование — это задача, которая для sympy уже не под силу. А вот человеку ее решить довольно просто, из чисто геометрических соображений:

```
vec4 cart_to_sh(vec4 pos)
{
    float r = sqrt(pos.x*pos.x + pos.y*pos.y + pos.z*pos.z);
    float th = PI / 2. - atan(pos.z/sqrt(pos.x*pos.x + pos.y*pos.y));
    float phi;
    if (pos.y == 0) {
        phi = sign(pos.x) * PI/2;
    } else if (pos.y >= 0) {
        phi = atan(pos.x/pos.y);
    } else {
        phi = PI + atan(pos.x/pos.y);
    }
    float t = pos.w;

    vec4 ret = {t, r, th, phi};
    return ret;
}
```

Обратите внимание на вычисление угла \phi. Дело в том, что при делении x/y теряется информация об исходных знаках переменных (++ неотличим от --, а

+- от -+). Без условного перехода пары точек слились бы в одну. Именно это мешает sympy вычислить обратное преобразование.

Преобразование координат векторов

Напомню, что геодезическая линия (т. е. наш луч света) — это путь перемещения вектора вдоль самого себя. А в криволинейных координатах один и тот же вектор, исходящий из двух разных точек, будет иметь разные компоненты. Для того чтобы преобразовывать координаты векторов из одной системы в другую, существует математический инструмент под названием матрица Якоби.

И тут на помощь снова приходит sympy:

```
J = coord_sh.jacobian(coord_cart)
IJ = simplify(J.inv())
print(f"J_{{spherical \\rightarrow cart}} = {latex(J)}")
print(f"J_{{cart \\rightarrow spherical}} = {latex(IJ)}")
```

```
J_{spherical \rightarrow cart} = \left[\begin{matrix}0 & \sin{\left(\phi \right)} \sin{\left(\theta \right)} & \rho \sin{\left(\phi \right)} \cos{\left(\theta \right)} & \rho \sin{\left(\theta \right)} \cos{\left(\phi \right)}\\0 & \sin{\left(\theta \right)} \cos{\left(\phi \right)} & \rho \cos{\left(\phi \right)} \cos{\left(\theta \right)} & - \rho \sin{\left(\phi \right)} \sin{\left(\theta \right)}\\0 & \cos{\left(\theta \right)} & - \rho \sin{\left(\theta \right)} & 0\\1 & 0 & 0 & 0\end{matrix}\right]J_{cart \rightarrow spherical} = \left[\begin{matrix}0 & 0 & 0 & 1\\\sin{\left(\phi \right)} \sin{\left(\theta \right)} & \sin{\left(\theta \right)} \cos{\left(\phi \right)} & \cos{\left(\theta \right)} & 0\\\frac{\sin{\left(\phi \right)} \cos{\left(\theta \right)}}{\rho} & \frac{\cos{\left(\phi \right)} \cos{\left(\theta \right)}}{\rho} & - \frac{\sin{\left(\theta \right)}}{\rho} & 0\\\frac{\cos{\left(\phi \right)}}{\rho \sin{\left(\theta \right)}} & - \frac{\sin{\left(\phi \right)}}{\rho \sin{\left(\theta \right)}} & 0 & 0\end{matrix}\right]
```

Теперь в любой точке, заданной в сферических координатах, мы можем получить матрицу, которая определяет трансформацию координат вектора с началом в этой точке.

Кроме точек, у которых \theta = 0. В них происходит деление на ноль. Но худшее что может случиться - артефакт в виде полоски, бьющей с полюсов черной дыры. Поэтому не будем слишком заморачиваться на этот счет.

При переносе на GLSL эти матрицы нужно транспонировать, так как они там хранятся в column-major формате:

```
// Value of Jacobi matrix at point sh_pos (point is given in Schwartschield coordinates)
// cartesian_vector = J * sh_vector
mat4 jacobian_inner_to_cart_at(vec4 sh_pos)
{
    float rho = sh_pos[1];
    float theta = sh_pos[2];
    float phi = sh_pos[3];

    // OpenGL stores matrices in column-major format. So this is a TRANSPOSED jacobian matrix
    mat4 ret = {
        {0, 0, 0, 1},
        {sin(phi)*sin(theta),      sin(theta)*cos(phi),      cos(theta),     0},
        {rho*sin(phi)*cos(theta),  rho*cos(phi)*cos(theta), -rho*sin(theta), 0},
        {rho*sin(theta)*cos(phi), -rho*sin(phi)*sin(theta),  0,              0},
    };
    return ret;
}

// Value of inverse Jacobi matrix at point sh_pos (point is given in Schwartschield coordinates)
// sh_vector = IJ * cartesian_vector
```

```
mat4 jacobian_cart_to_inner_at(vec4 sh_pos)
{
    float rho = sh_pos[1];
    float theta = sh_pos[2];
    float phi = sh_pos[3];

    // OpenGL stores matrices in column-major format. So this is a TRANSPOSED jacobian matrix
    mat4 ret = {
        {0, sin(theta)*sin(phi), sin(phi)*cos(theta)/rho,  cos(phi)/(rho*sin(theta))},
        {0, sin(theta)*cos(phi), cos(phi)*cos(theta)/rho, -sin(phi)/(rho*sin(theta))},
        {0, cos(theta),          -sin(theta)/rho,          0},
        {1, 0, 0, 0},
    };
    return ret;
}
```

Рисуем остаток совы

Поверхность сферы прямо над горизонтом событий. При взгляде с одной стороны видно всю сферу сразу.

Поверхность сферы прямо над горизонтом событий. При взгляде с одной стороны видно всю сферу сразу.

Итак, наш фрагментный шейдер будет вызван для каждого пикселя в области отрисовки. Результат его работы — цвет этого пикселя.

```
layout(location = 0) out vec4 diffuseColor;

#define PI 3.14159265358979323846264338327795

uniform vec2 window_size;
uniform vec4 camera_pos;
```

uniform mat3 camera_rotation;

uniform float viewport_dist = 1.0;

uniform float rs = 0.25;

uniform float shell_radius = 2.0;

На вход он принимает следующие uniform (т.е. общие для всех пикселей) переменные:

размер области отрисовки

параметры камеры (позицию в сферических координатах, матрицу поворота в декартовых, расстояние до точки схождения лучей)

радиус Шварцшильда

радиус оболочки

Координаты самого пикселя доступны во встроенной переменной gl_FragCoord.

Осталось самое простое:

Отправляем луч света через пиксель назад во времени

В цикле двигаем его по геодезической с помощью rk34_step

При контакте с оболочкой, красим пиксель в её цвет. Если он при этом еще и покидает оболочку - выходим.

Поехали!

void main()

Результат:

Получилось на самом деле довольно-таки неоптимально. Деление на почти-ноль около полюсов сильно портит производительность, заставляя интегратор выбирать очень маленькие шаги для сохранения точности. Только посмотрите на это зловещее красное свечение: оно пропорционально количеству итераций, потраченному на каждый из пикселей!

А это - максимальная оценка ошибки интегратором:

Кстати, все остальные картинки в этой статье (кроме тора) я получил просто добавив на эту сферу заливку другими цветами и подкрутив ее радиус.

Думаю, можно оптимизировать это всё еще в пару десятков раз. Но это я оставлю в качестве упражнения читателю.