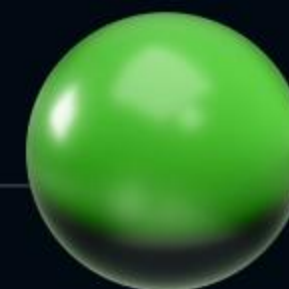


# Демосцена: в погоне за wow-фактором




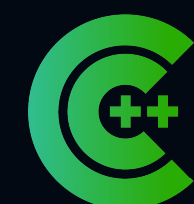
**Александр  
Кухаренко**

WANNA



 @lastshilling

 0f0x64@gmail.com



**C++ Russia**  
2023

**WANNA**

# Биография



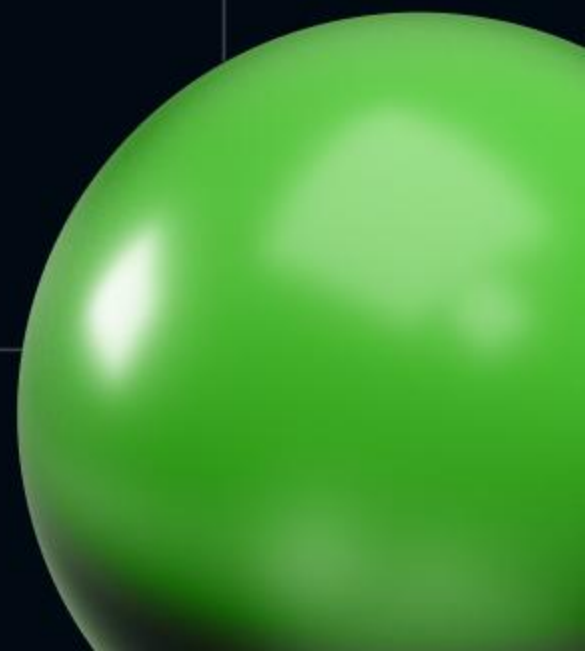
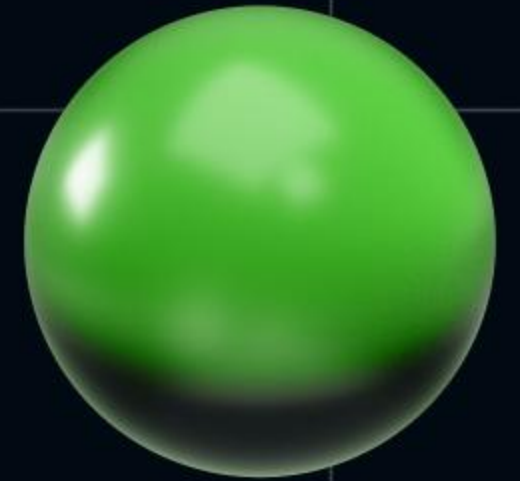
**Александр  
Кухаренко**

✈ @lastshilling

- занимаюсь компьютерной графикой и пишу демо с 1997 года
- работаю рендер-инженером (фотореалистичная графика, дополненная реальность) в компании WANNA
- пишу и исполняю музыку - электронную (с 2000) и живую (с 2007-го)

# План доклада

- что такое демосцена
- технологический фронтир. старая и новая школы
- три кита: абстрактное искусство, процедурная генерация, сайзкодинг
- собираем 64к интро: чистка мусора и сайзкодинг
- собираем 64к интро: генерация ассетов и сцен
- собираем 64к интро: генерация звука и музыки
- ui для демо-движка
- как использовать все это вне демосцены?





# Что такое демосцена?

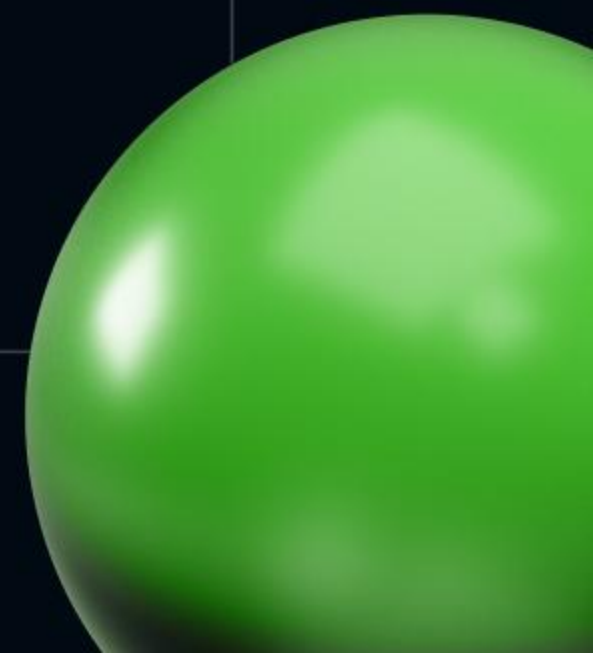
256 байт



4 килобайта



64 килобайта

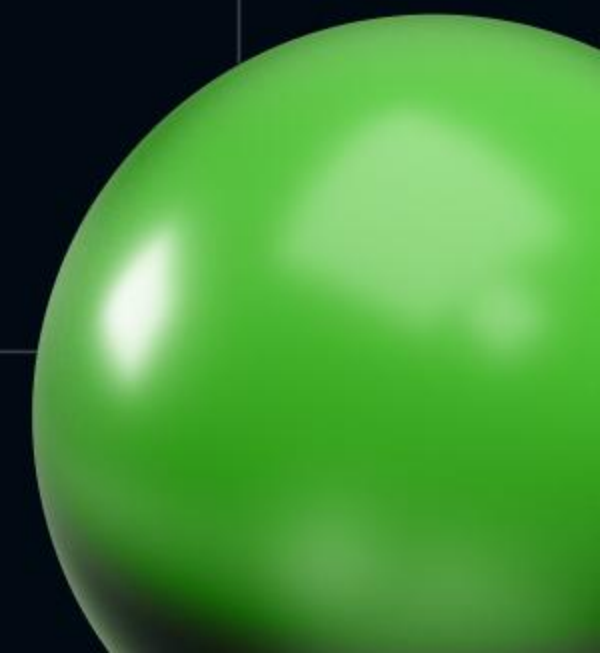


# Зачем это всё?

- “потому что могу”
- показать себя, впечатлив людей
- создать/опробовать новую технологию
- для саморазвития или развлечения (демо как побочный продукт)

Как образ жизни:

- демосцена это и спорт, и искусство, и субкультура





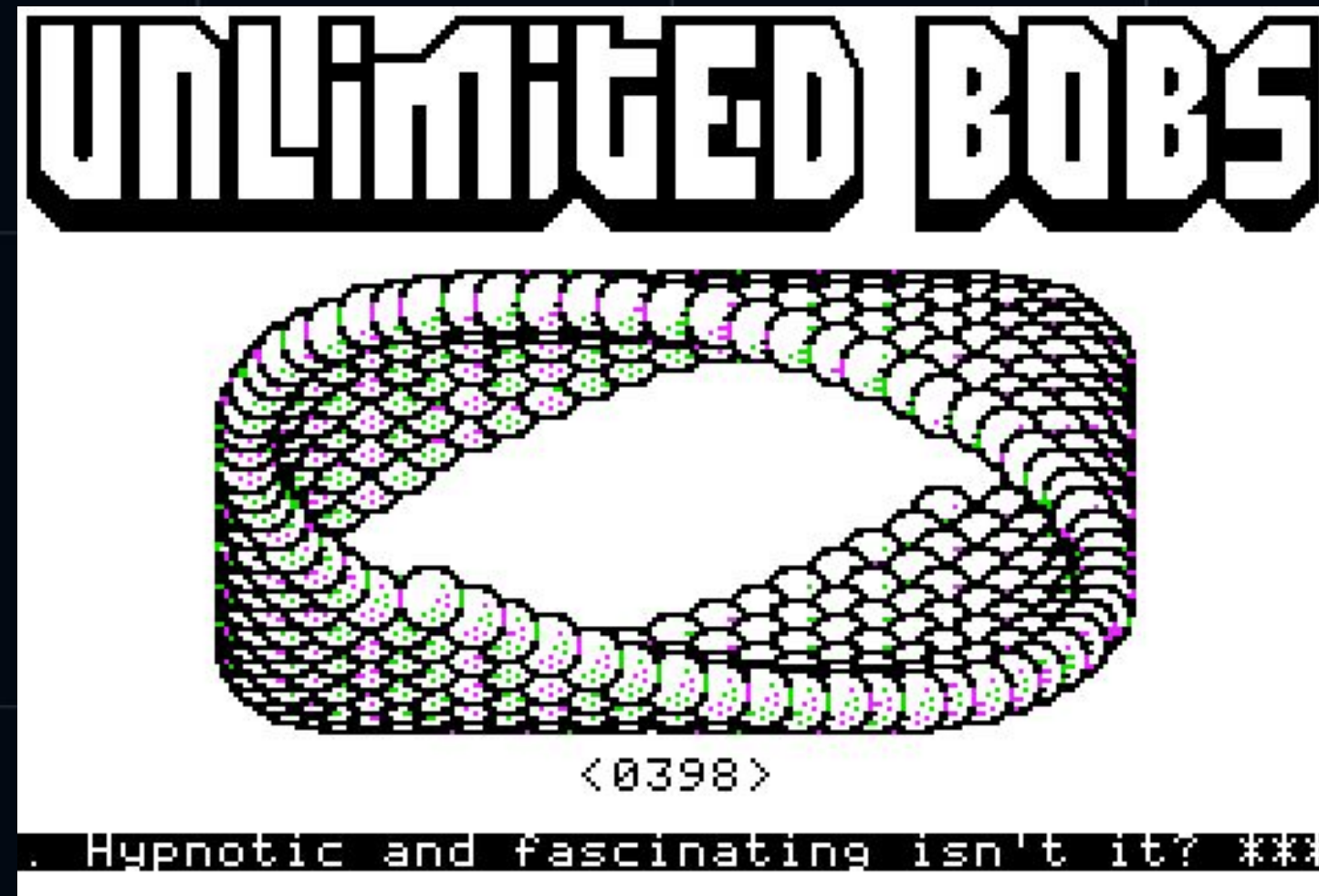
# Технологический фронтир

Железо старой школы:

2-16 цветов,  
простейшие звуковые чипы,  
слабые процессоры, 2д графика

Демосцена старой школы:

256 цветов  
цифровая музыка  
3d графика, “невозможные” эффекты



циклично проигрываемый  
анимационный буфер,  
каждый фрейм в него рисуется один шарик

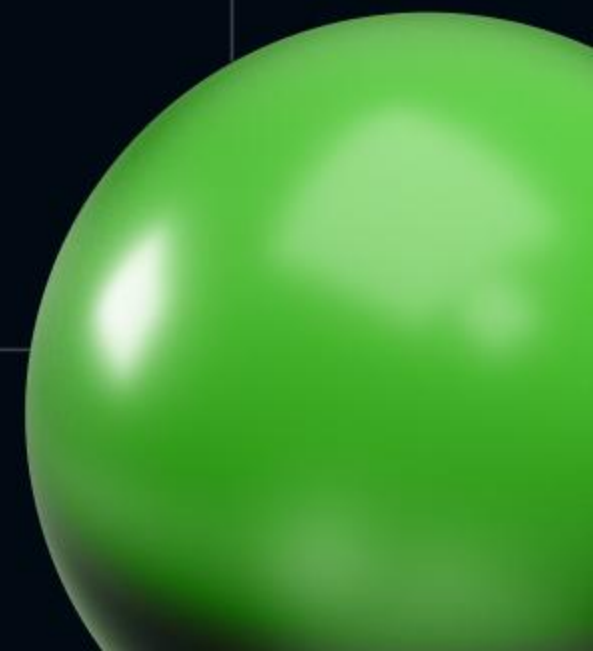
# Технологический фронтир

Железо новой школы:

полная цветовая палитра, звук hiRes, 3d близкое к фотореализму

Демосцена новой школы:

?





# Демосцена как контркультура

демонстрация мощи “железа”,

фотореализм

-> абстрактное искусство

работа художника, моделлера,

композитора

-> процедурная генерация

отсутствие ограничений на память

-> сайзкодинг

NB! Если задача решена технологически -

работа программиста закончена,

начинается работа художника.

отсутствие коммерческой составляющей развязывает руки  
в выборе технологий



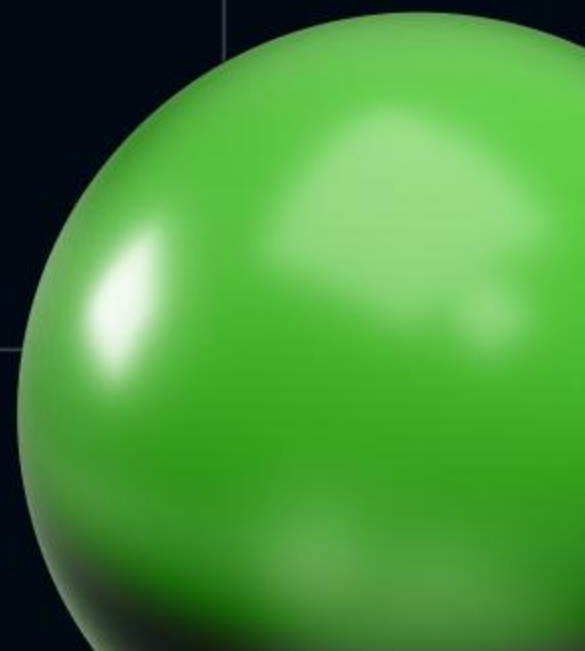
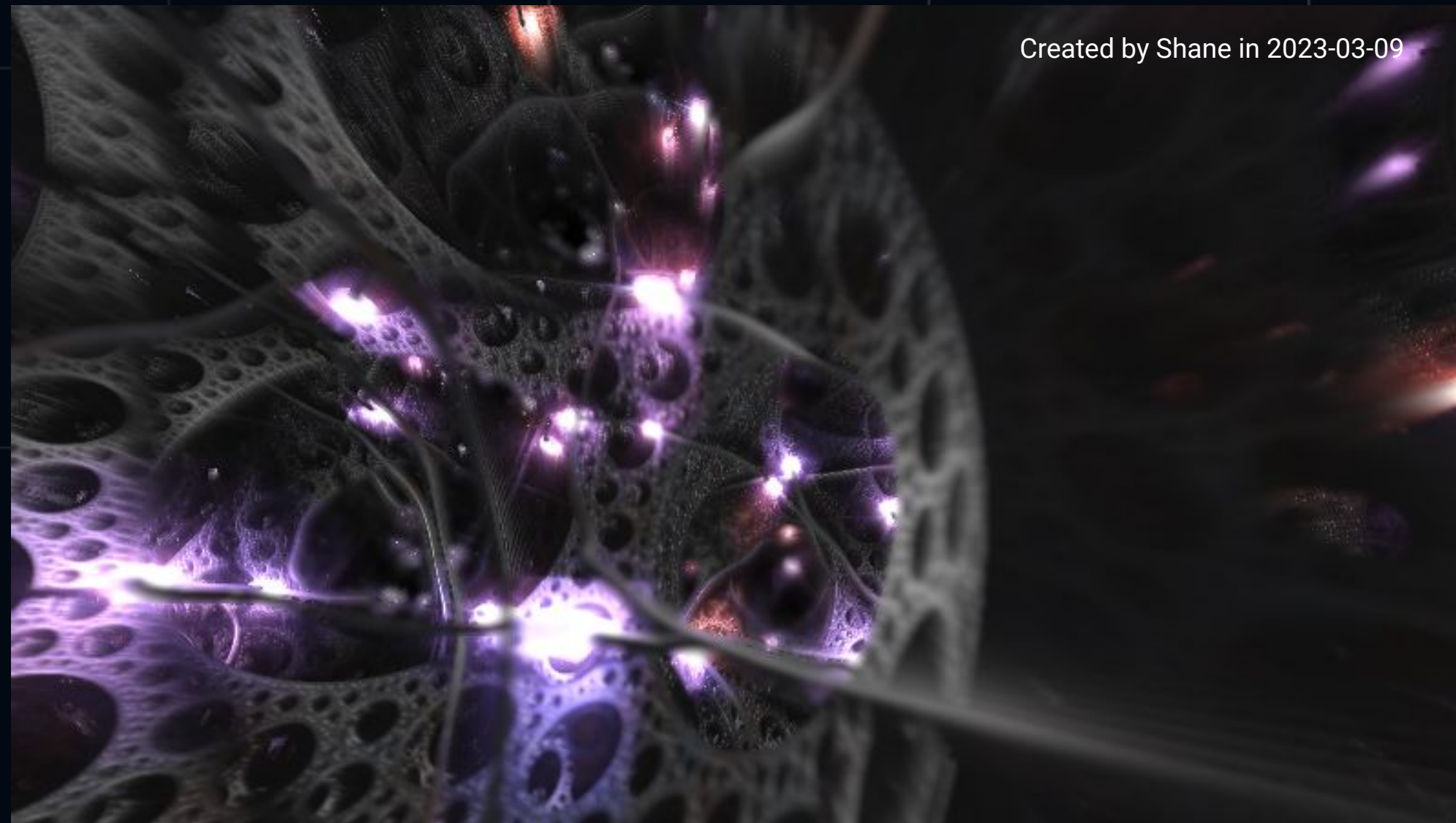
На самолете проще и быстрее -  
но мы все равно устраиваем гонки на автомобилях



# Абстрактное искусство

визуализация  
математических  
формул

sdf  
фракталы  
реймаршинг





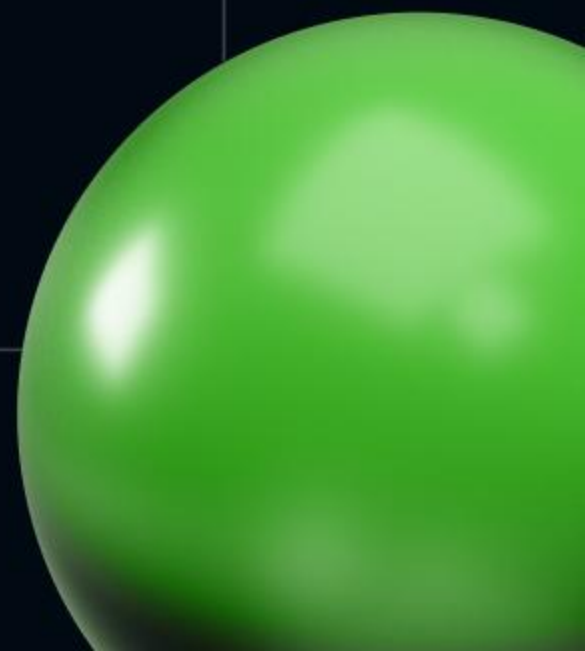
# Процедурная генерация



симуляция воды -  
одна из первых “коммерческих”  
задач генеративной графики



генерировать ландшафты процедурно  
зачастую гораздо быстрее,  
чем создавать “руками”





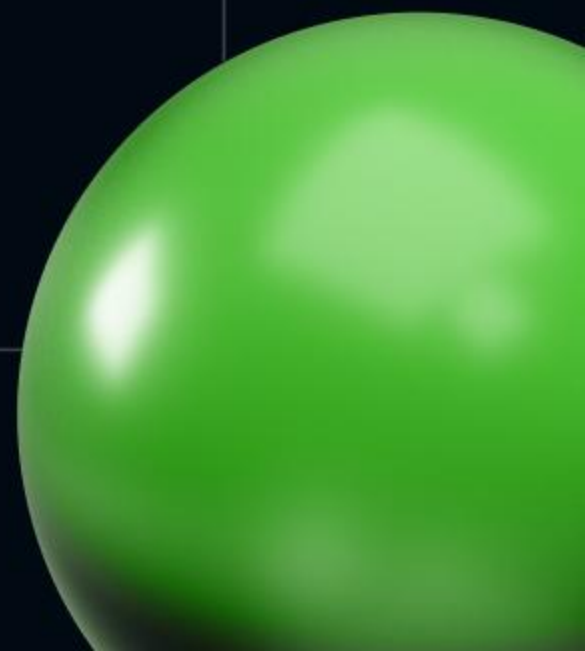
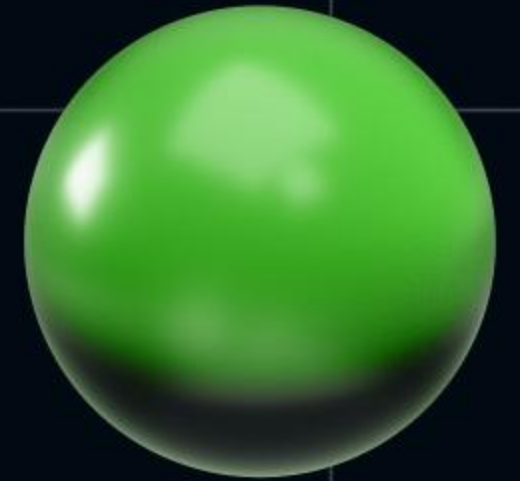
# Сайзкодинг

сожмём в 64кб все что угодно?



как это вообще возможно?

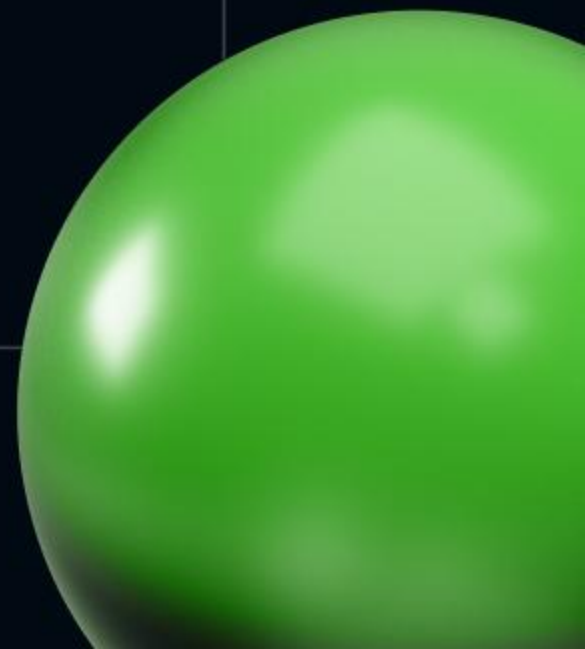
1. избыточность  
информации реального  
мира (повторяемость)
2. схожесть математики  
с реальным миром  
(заменяемость)





# Сайзкодинг

- уберем из .exe [лишнее](#)
  - из коробки в visual studio мы получим .exe размером в 70-160кб
  - удаляем все ресурсы, линкуем crt динамически - размер .exe ~= 11кб
  - не используем crt вообще - размер .exe ~= 3кб
  - для тех кому некогда - есть фреймворки [1к-4к](#) [64к](#)
  - аналогично все происходит и под [линуксом](#)



# Сайзкодинг



не используем сторонние библиотеки

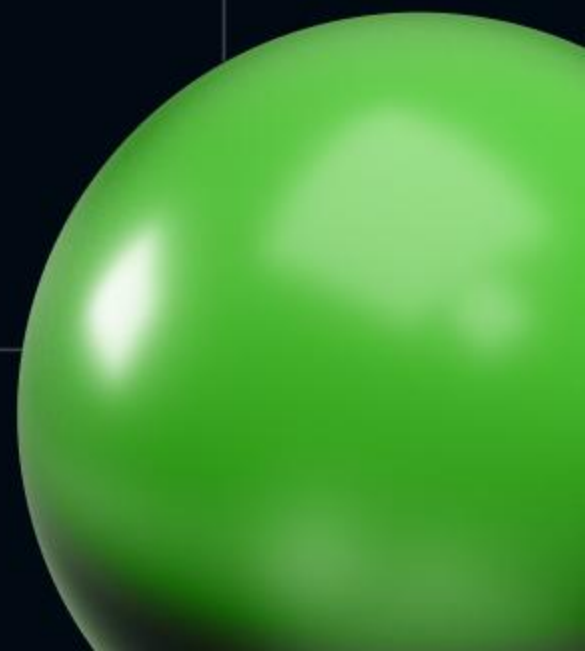
(не знаешь, во что скомпилируется - за борт!)

выкинув CRT, также забываем про RTTI, исключения, STL

пакуем все с помощью [kkrunchy](#) или [crlinkler](#)

футпринт такого фреймворка оказывается около 6кб в случае kkrunchy и около 600

байт в случае использования тул для 4k



# Генерация ассетов

- проанализируем нашу сцену и выделим наиболее “прожорливые” участки
  - обычно для графики это текстуры, а для звука - семплы
  - следующие по списку - геометрия и нотный текст
- напомним основные генераторы и фильтры
  - perlin noise, cells, blur, equalizer, fft, dla, l-system...
  - агрессивно оптимизируем их код по размеру  
(частый трейдофф - оперативная память, реже - скорость)



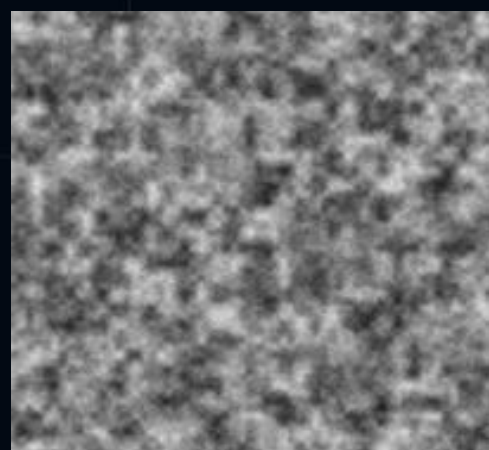


# Генерация ассетов

Создадим объекты последовательно вызывая генераторы и фильтры с разными параметрами. Множество объектов реального мира можно симулировать используя комбинацию шумов, фракталов и простой геометрии.



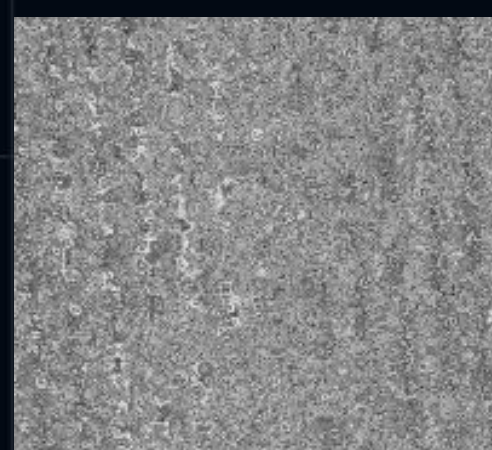
в “чистом виде” (без обработки) генераторы смотрятся плохо



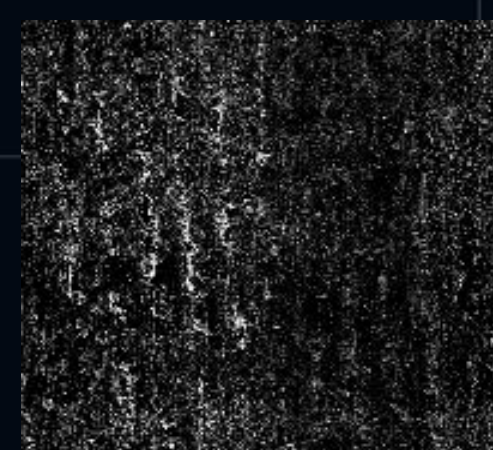
perlin noise



perlin c  $y \cdot \text{const}$



perlin(perlin)



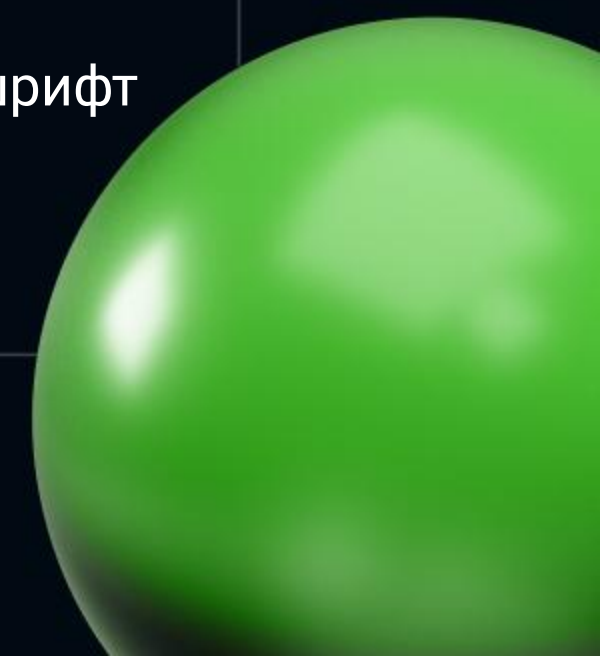
perlin(perlin(perlin))  
+contrast



sdf

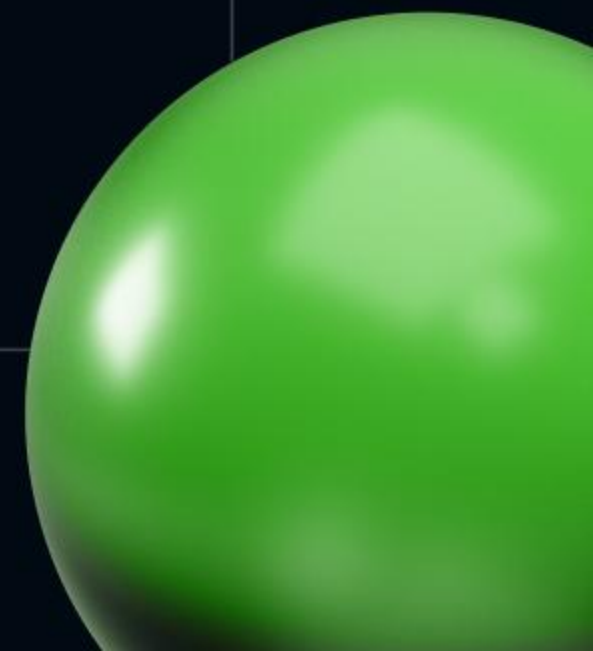


2 sdf + шрифт



# Генерация ассетов

Смешивание. Алгоритм выбирается под задачу,  
для достижения нужного эффекта



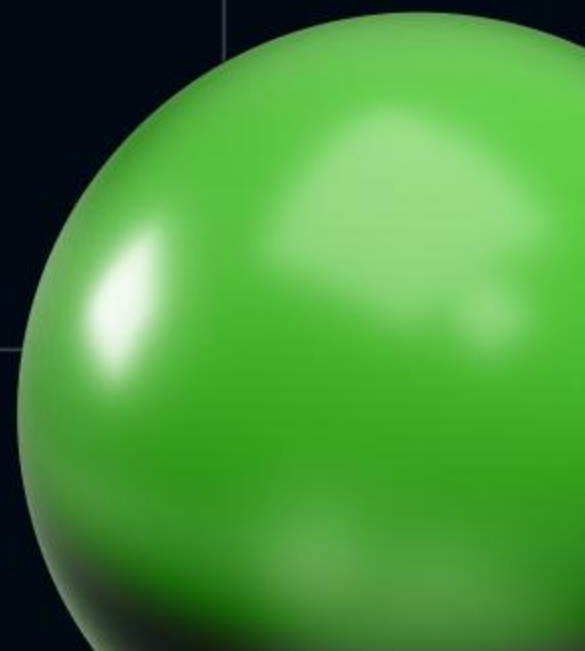
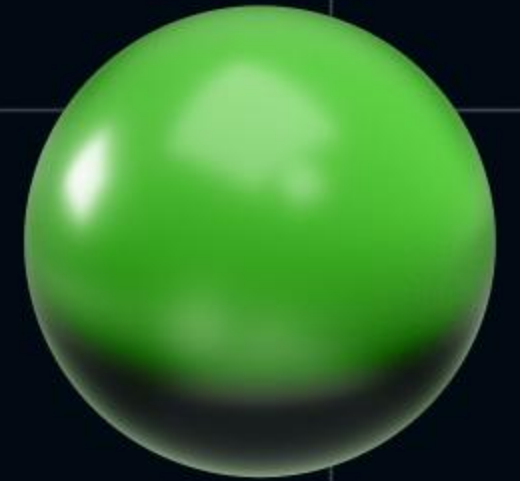


# Генераторы

```
float hash(float n) { return frac(sin(n) * 43758.5453); }
```

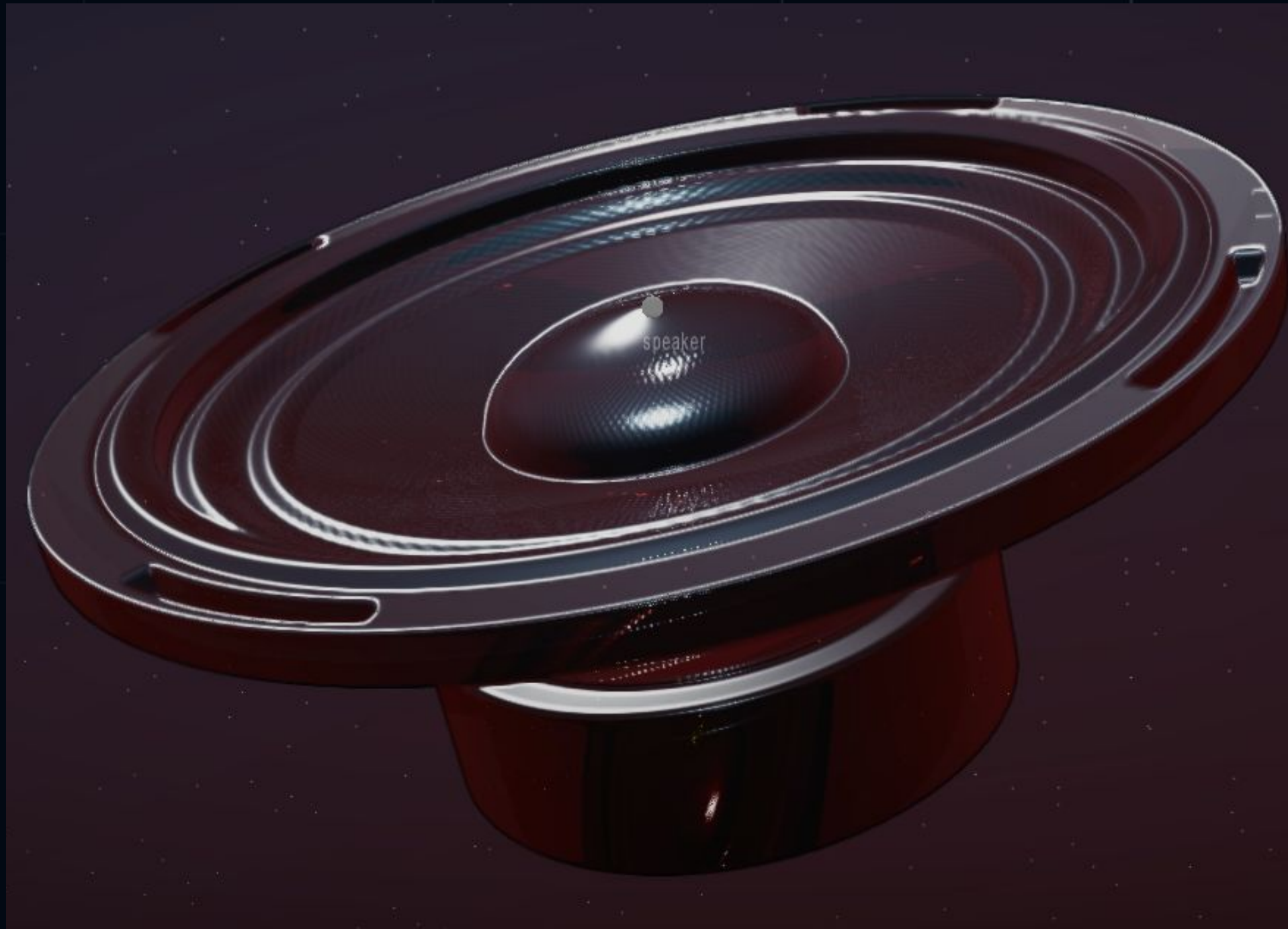
```
float noise(float2 uv) {  
    float3 x = float3(uv, 0); float3 p = floor(x); float3 f = frac(x); f = f * f * (3.0 - 2.0 * f);  
    float n = p.x + p.y * 57.0 + 113.0 * p.z;  
    return lerp(lerp(lerp( hash(n + 0.0), hash(n + 1.0), f.x), lerp( hash(n + 57.0), hash(n + 58.0), f.x), f.y),  
        lerp(lerp( hash(n + 113.0), hash(n + 114.0), f.x), lerp( hash(n + 170.0), hash(n + 171.0), f.x), f.y), f.z);  
}
```

```
float perlin(float2 uv) {  
    float res = noise(uv) * 64.0 + noise(uv * 2.0 ) * 32.0 + noise(uv * 4.0 ) * 16.0 + noise(uv * 8.0 ) * 8.0;  
    return res / (1.0 + 2.0 + 4.0 + 8.0 + 16.0 + 32.0 + 64.0) - 0.5;  
}
```

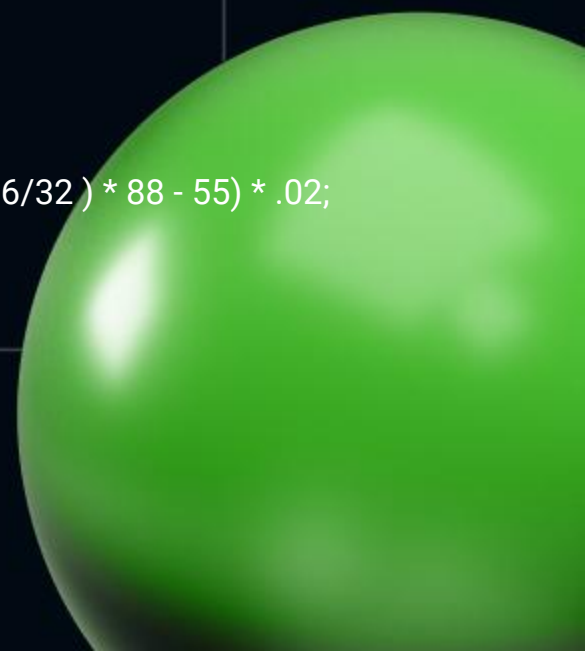




# Генераторы (геометрия)



```
float4 PS(VS_OUTPUT input) : SV_Target
{
    float2 uv=input.uv - .5;
    float2 a= uv * PI * 2;
    a *= 1 - float2(sectorU, sectorV) / 360;
    a += float2(offsetU, offsetV) / 360 * PI * 2;
    float3 pos = float3 (sin(a.x), .5 * (a.y / 2), cos(a.x));
    pos *= - .5;
    if (uv.y < 0) {pos.xz *= 1 + uv.y * 2; pos.y = uv.y / 4;}
    if (uv.y < 0 && uv.y > - .1) { pos.y=0; }
    float r = 256 * 3.;
    if (uv.y < - 0.05 && uv.y > -.2) { pos.y += -sin(uv.y * 111) * .0051; }
    if (uv.y<0.1) pos.y += .0005*sin(uv.x * r) * sin(uv.y * r) * pow( saturate( -(uv.y-.3) + .15), .2);
    float m = .35;
    if (uv.y < -m) pos.y += pow( ( -uv.y - m), .5) * .3;
    if (uv.y > 0.05 && uv.y<.3) { pos.xz *= saturate( 1 - uv.y * 2) + .015; }
    if (uv.y >= 0.3) { pos.xz *= .5; }
    m=.49;
    if ( uv.y > m ) { pos.xz = pos2.xz * ((uv.y - m)/m); }
    if ( uv.y< -.015 && uv.y > -.035) pos.y -= saturate( sin(PI * uv.x * 256/32 ) * 88 - 55) * .02;
    return float4(pos.xyz, 1);
};
```



# Генераторы (геометрия)



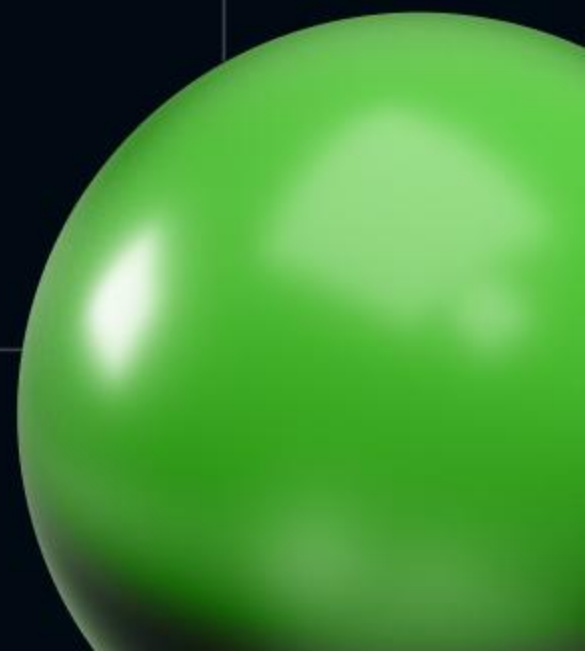
```
float2 uv=input.uv - .5;  
float2 a= uv * PI * 2;  
a *= 1 - float2(sectorU, sectorV) / 360;  
a += float2(offsetU, offsetV) / 360 * PI * 2;  
float3 pos = float3 (sin(a.x), .25 * a.y, cos(a.x));  
pos *= - .5;
```



```
if (uv.y < 0)  
{  
    pos.xz *= 1 + uv.y * 2;  
    pos.y = uv.y / 4;  
}
```



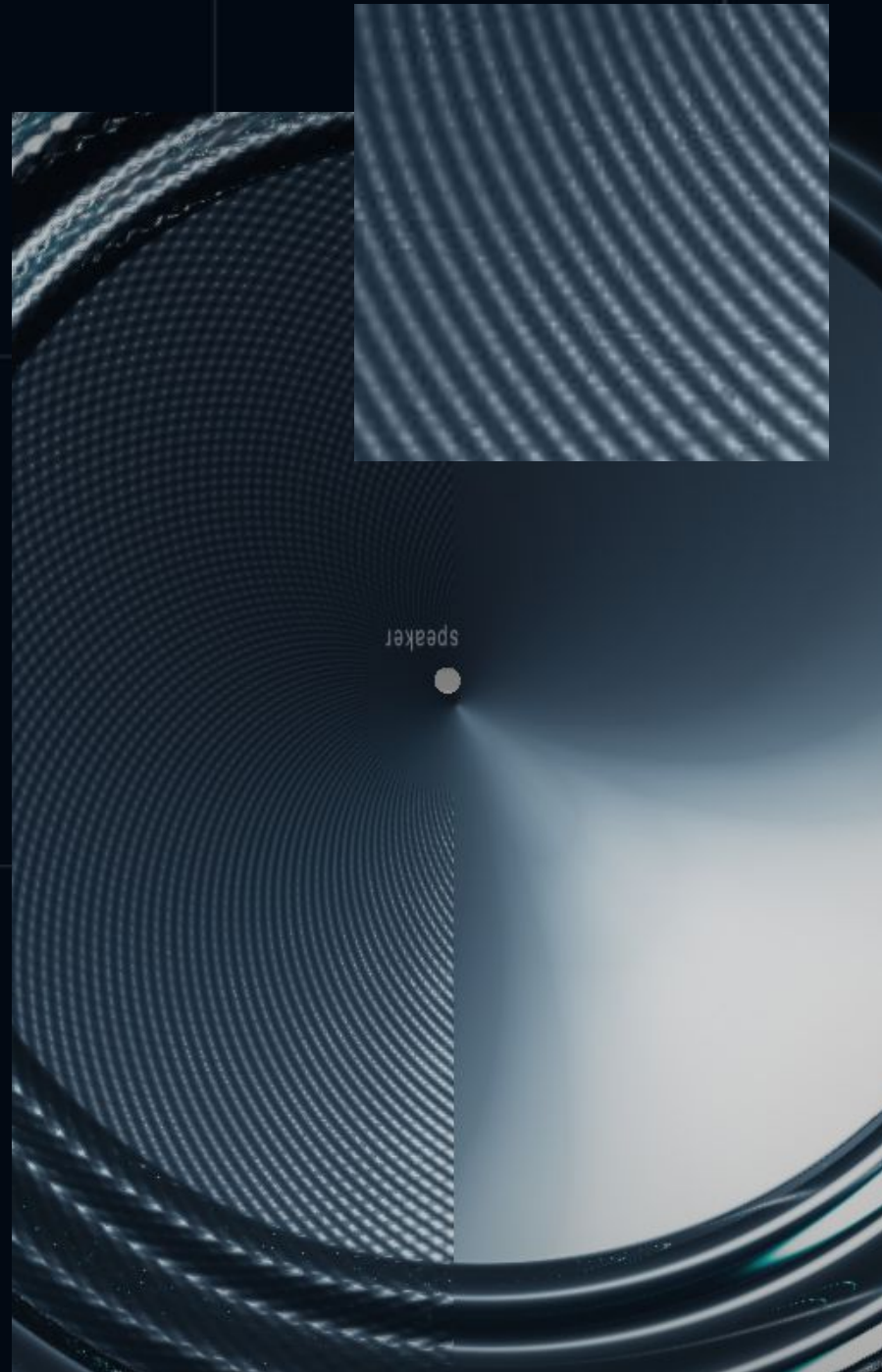
```
if (uv.y < 0 && uv.y > - .1)  
    pos.y=0;
```





# Генераторы (геометрия)

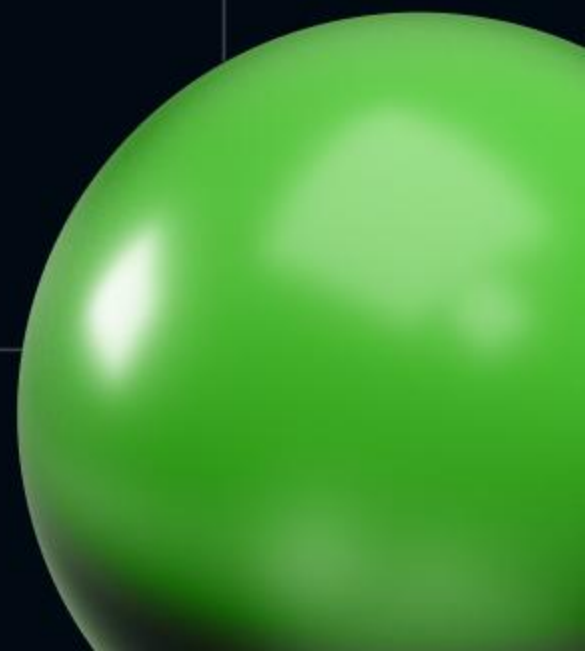
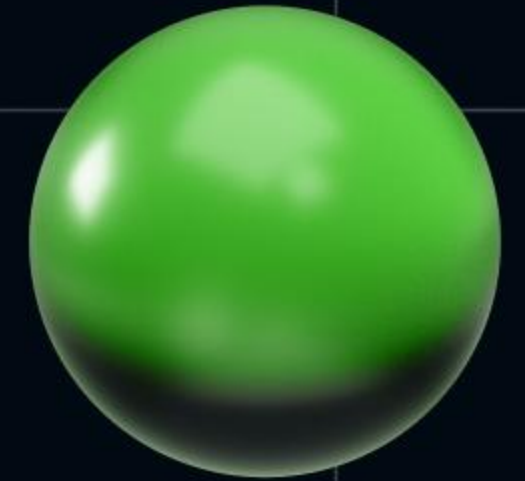
```
float r = 256 * 3.;  
if (uv.y < - 0.05 && uv.y > - .2)  
    pos.y += -sin(uv.y * 111) * .0051;
```



```
if (uv.y < -0.1)  
    pos.y += .0005 *  
        sin(uv.x * r) *  
        sin(uv.y * r) *  
        pow( saturate( -(uv.y-.3) + .15), .2);
```



```
if (uv.y > 0.05 && uv.y < .3)  
    pos.xz *= saturate( 1 - uv.y * 2) + .015;  
if (uv.y >= 0.3)  
    pos.xz *= .5;
```

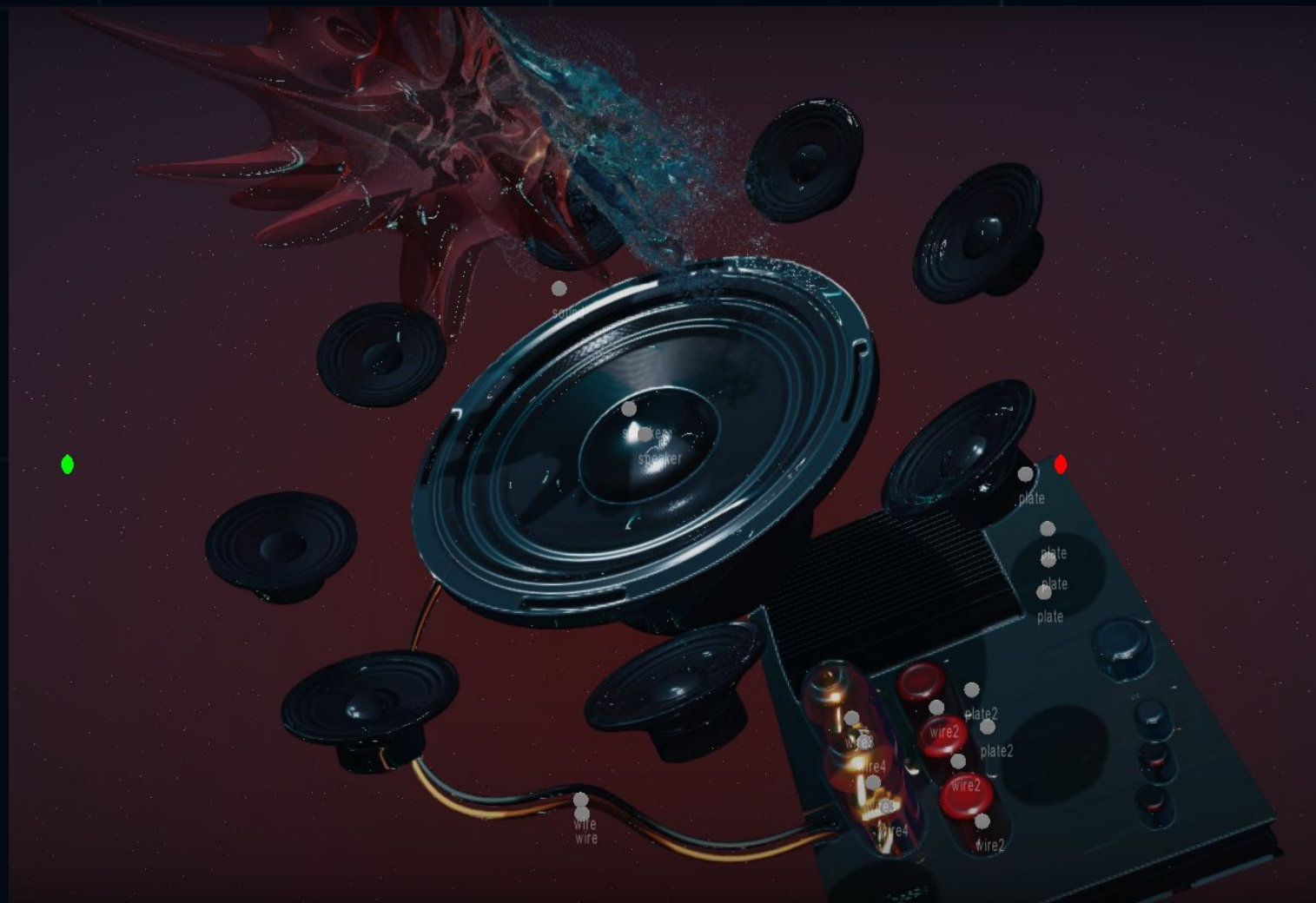




# Генерация сцены

Собираем всё вместе. Если количество объектов невелико - расставим их вручную, иначе - используем программные скаттеры.

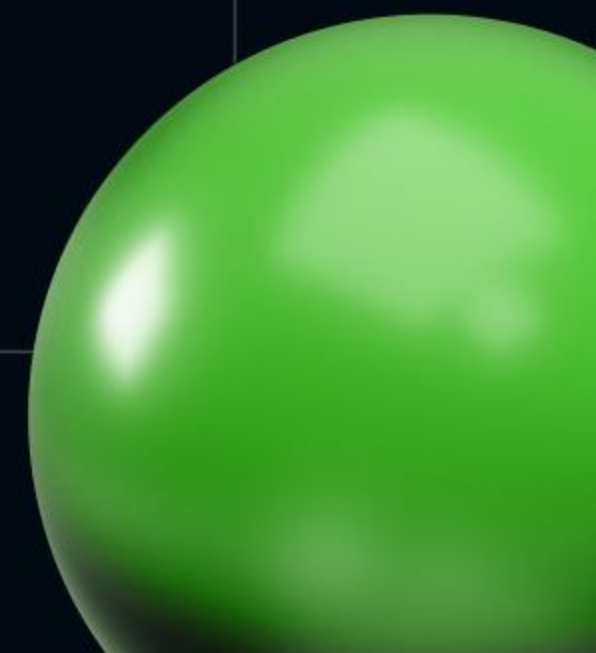
Скаттеры можно базировать на тех же алгоритмах, что и генераторы.





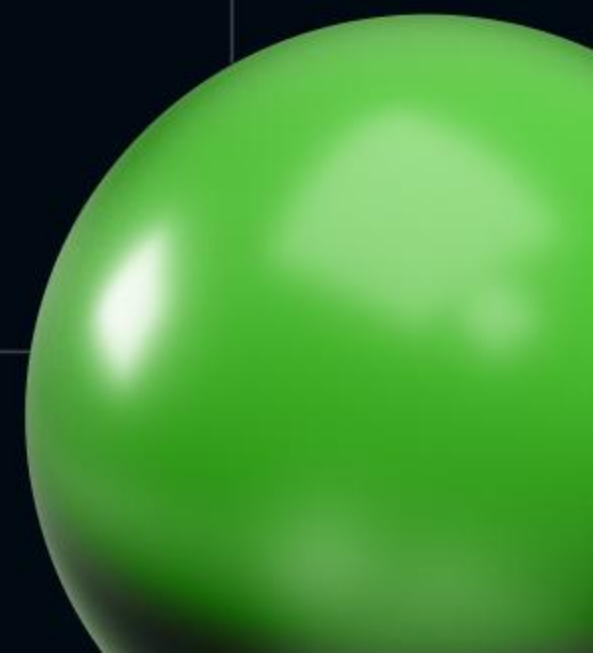
# Генерация сцены

В расположении объектов всегда скрыта система -  
или человеческая логика, либо природный закон



# Сайзкодинг

- напомним простую виртуальную машину (интерпретатор байткода)
  - реализация через абстрактный класс и виртуальные методы
  - либо можно использовать указатели на функции
- запишем нашу сцену как скрипт
  - следим за размерностью параметров: где нужен char, где short, где float
- большой массив параметров (например, геометрию в векторном виде) можно сжать с помощью дельта кодирования. Также, возможно применение сжатия с потерями
- по шейдерному коду пройдемся [минифаером](#)
- соберем .exe и не забудем про [exe-пакер](#)



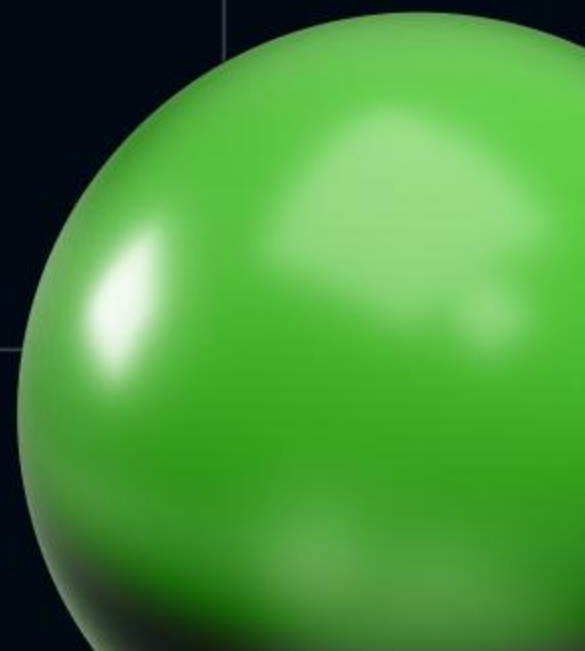


# Звук

Типичная схема генерации звука

[нотный текст -> генератор волны -> фильтр]

- современная музыка как правило, это или loop based, или ambient. Это хорошо для сжатия - нотный текст можно не генерировать, а хранить.
- современная музыка по большей части - электронная. Значит, если взять те же алгоритмы, что в синтезаторах, и звучать будет похоже.
- с “живыми” инструментами все сложнее, поскольку сложнее и сама музыка.
- анализ звучания строится, в основном, на изучении АЧХ и формы волны.



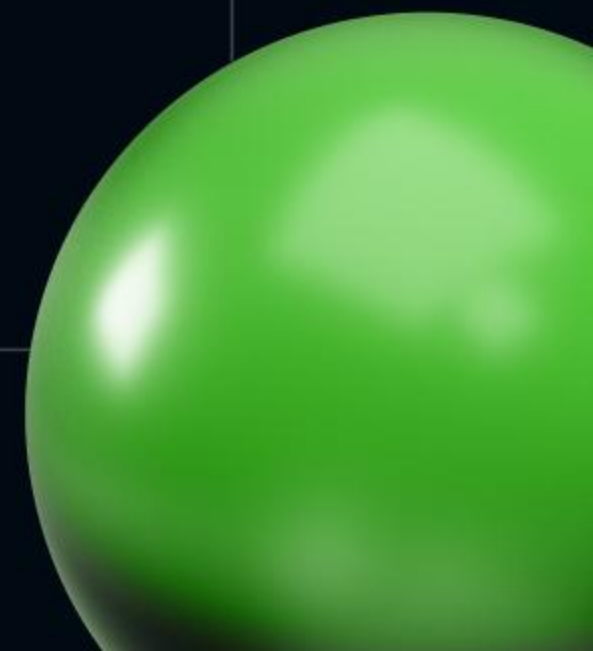
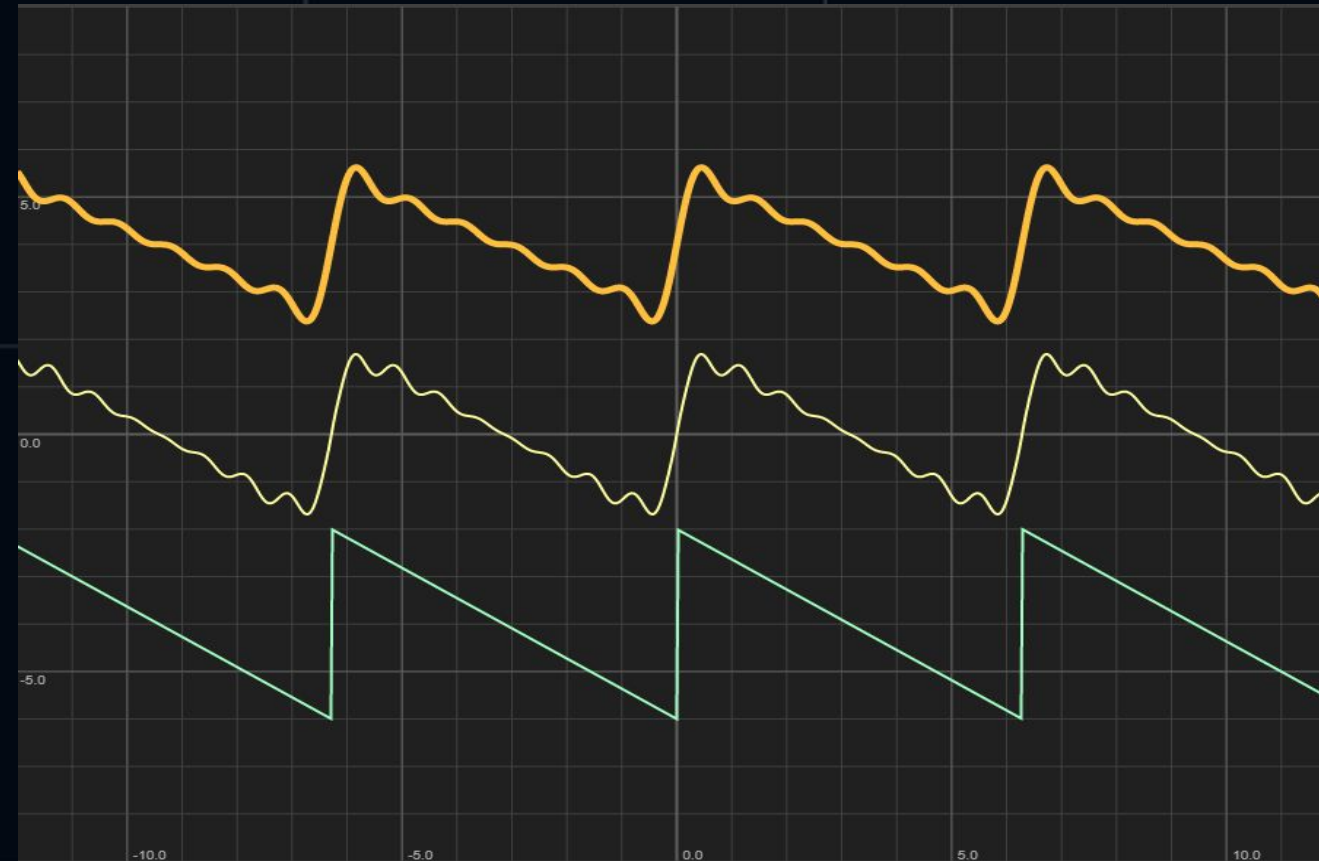
# Генерация волны

Три варианта генерации волны типа "пила"

```
1. float h = 6;  
for (float g = 1; g <= h; g++) {  
    a += sin(x * g) / g;  
}
```

```
2. float h = 6;  
a = (2 * lerp (frac( -x / (PI * 2)), .5, pow(1 - abs(sin( -x / 2.)), h)) - 1) * (1 - cos(x*h) /h. );
```

```
3. a = 1 - frac(x);
```



# Фильтрация звука

Простейший фильтр НЧ

```
a[i] = lerp(a[i], a[i+1], 1 - weight);
```

Фильтр задержки можно собрать вот так

```
a[i] = lerp(a[i], a[i-delay], mix);
```

Простейший дисторшн

```
a[i] = clamp(a[i] * amp, -1, 1);
```

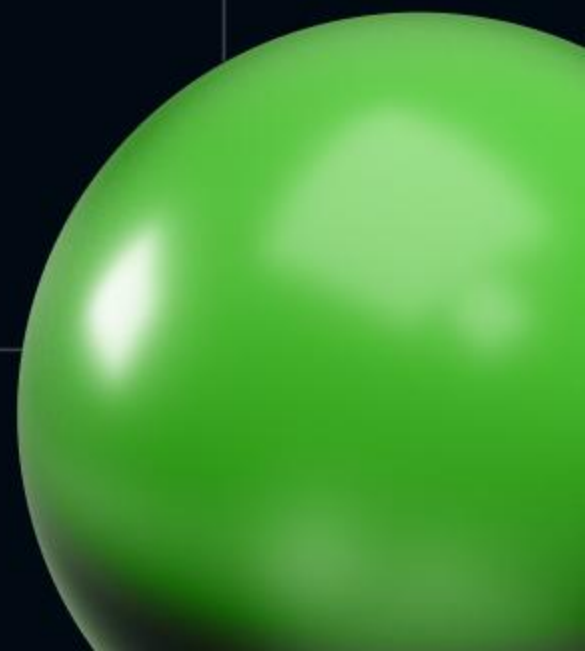
Написать фильтр несложно. Сложно написать фильтр с заданными характеристиками!





# Грязные трюки

- спич-синт windows
- ограбление gm.dls
- чтение системных шрифтов



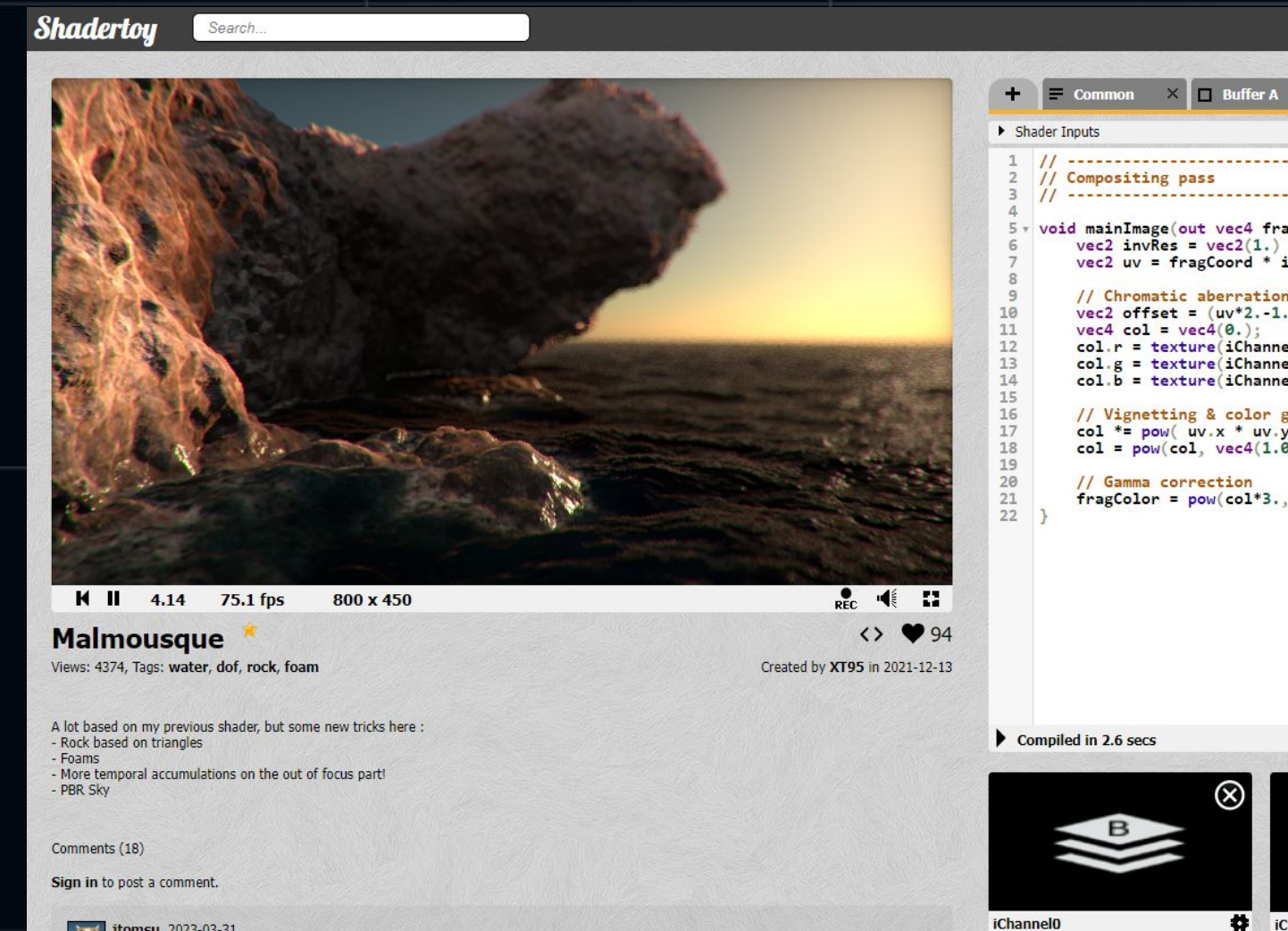
# UI



Почему штатной среды разработки недостаточно?

Важность визуальной настройки параметров: интерактивность = скорость

- подход [shadertoy.com](https://shadertoy.com)  
минималистичный интерфейс (мир - это текст)  
+ можно очень быстро написать  
- работать в нем - очень медленно!
- UE-like подход  
все в точности наоборот

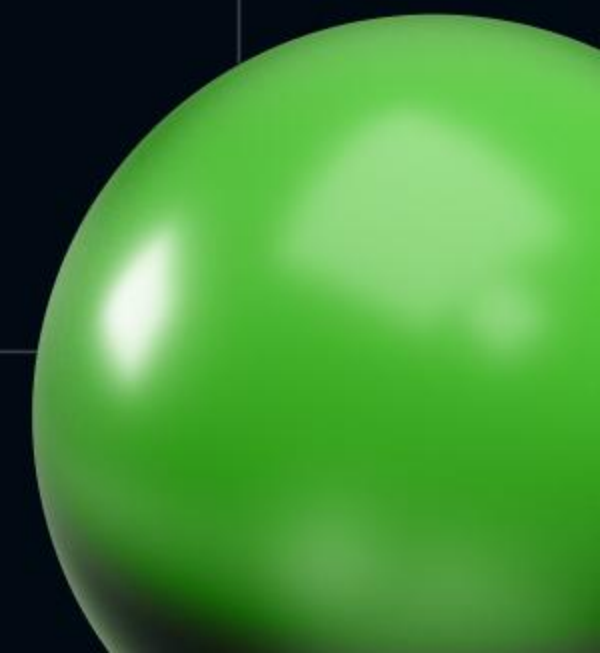




# UI

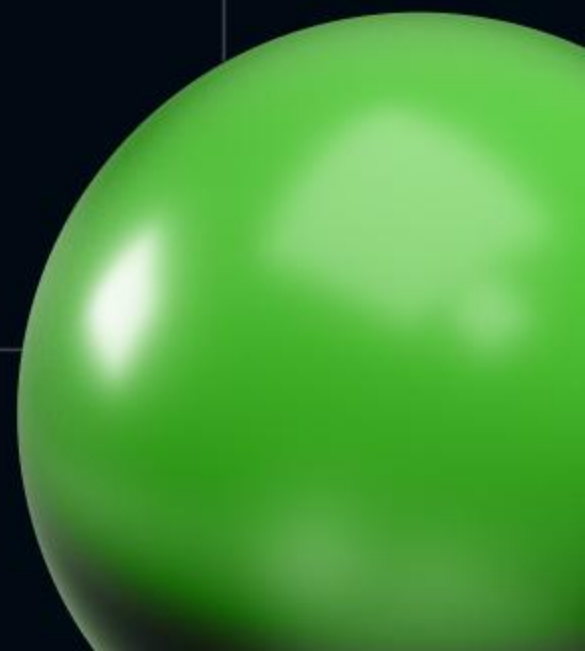
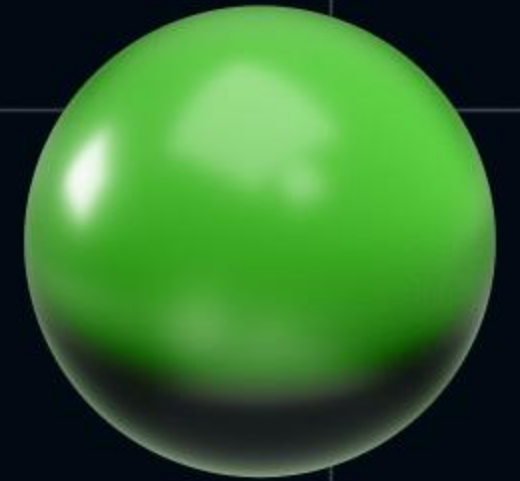
Один из главных параметров движка (включая UI) - скорость внесения изменений.

Это значит - [интроспекция и рефлексия](#) обязательны



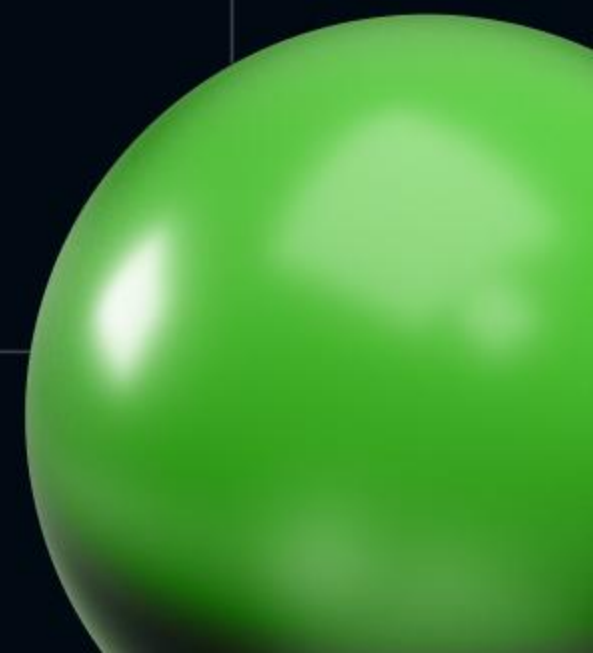
# Как использовать вовне?

- в рабочем процессе:
  - песочница для прототипирования  
(небольшой движок, быстрая сборка, гибкость)
- в качестве портфолио
- в продакшне:
  - бесконечные ассеты
  - ускорение производства контента
  - ускорение модификации контента





# Вопросы?



# Спасибо!

главный международный ресурс о демосcene:

<https://www.pouet.net/>

русскоязычный чат о демосcene:

<https://t.me/pcdemomaking>

вопросы по моим наработкам можно задать здесь:

<https://t.me/fxEngineChat>

русскоязычный чат по генеративному арту:

[https://t.me/gen\\_c](https://t.me/gen_c)

