Вперед в прошлое, или Разрабатываем фреймворк под Windows 95 в 2023 году

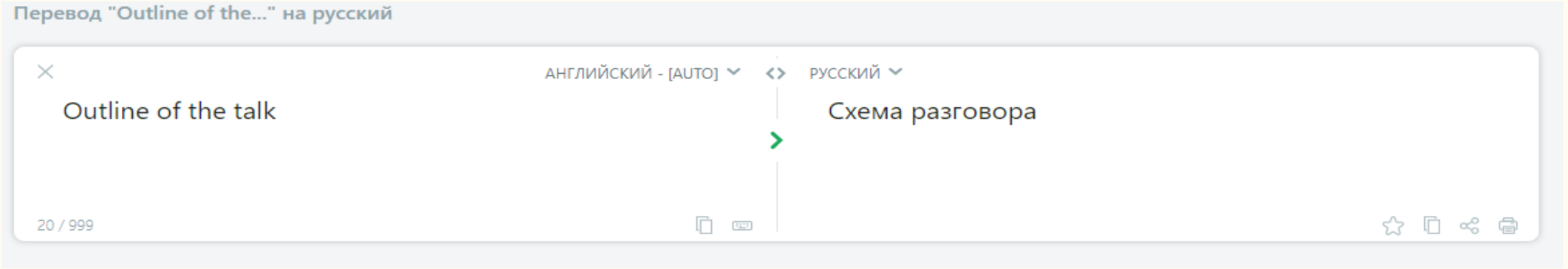
Евгений Зощук

Ренессанс банк

User not found

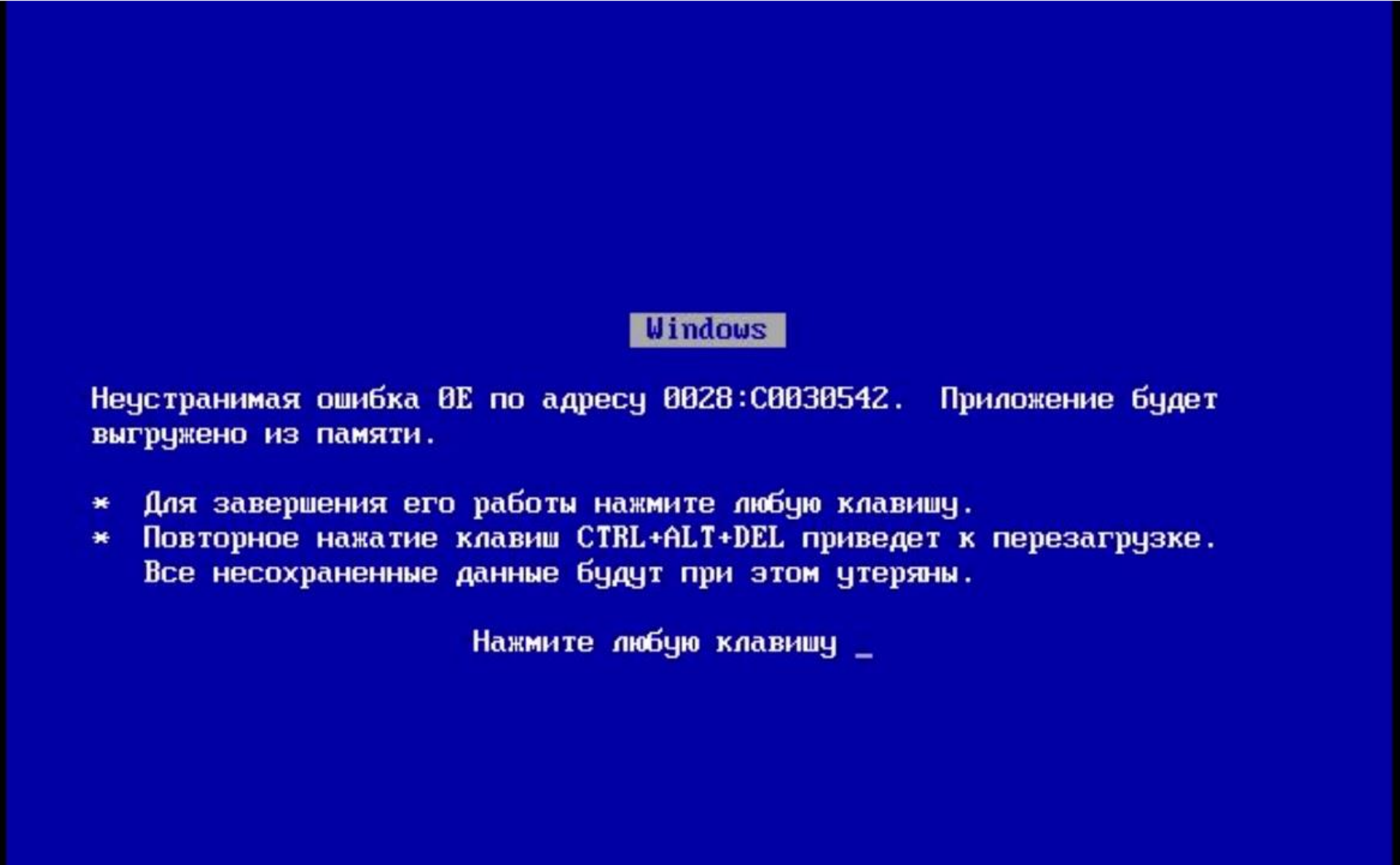
Jordan631@Yandex.ru

Outline of the talk



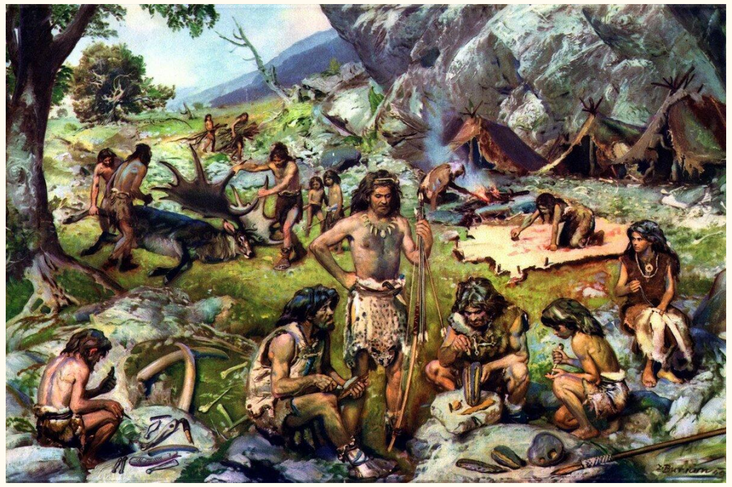
1.0f Окунемся в историю ПК 90-ых
2.0f Вспомним как разрабатывали бородачи
3.0f Сравним железо разных лет
4.0f Посмотрим насколько увеличилась производительность
5.0f Узнаем как писать фреймворк в 2023

```
for (int i = 0; i < sizeof(titles) / sizeof(titles[0]); i++)
{
    printf("%d - %s", i, titles[i]);
}
```

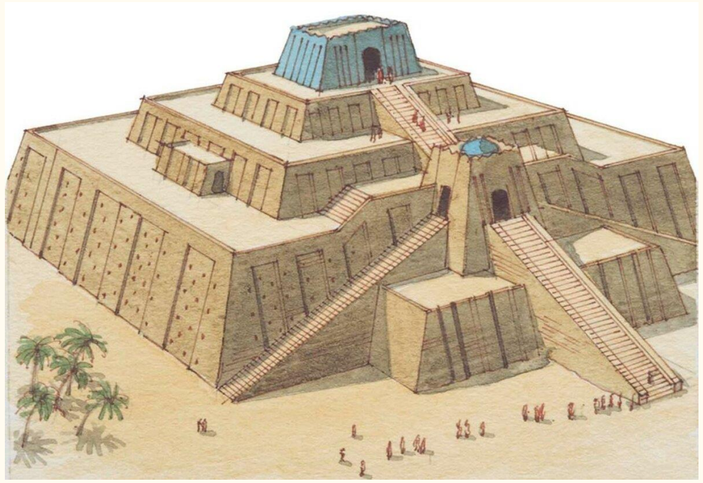

Windows
Неустранимая ошибка 0E по адресу 0028:C0030542. Приложение будет выгружено из памяти.
* Для завершения его работы нажмите любую клавишу.
* Повторное нажатие клавиш CTRL+ALT+DEL приведет к перезагрузке. Все несохраненные данные будут при этом утеряны.
Нажмите любую клавишу _

Краткая история компьютеров

Нужно изобрести транзистор

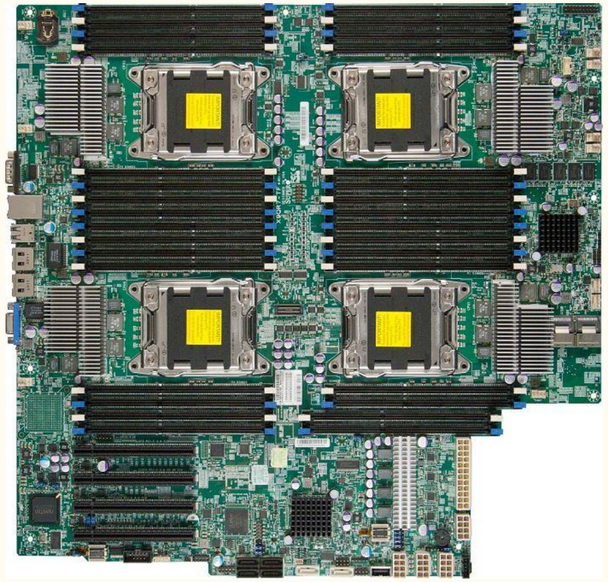


Нужно больше ~~зикуратов~~ транзисторов

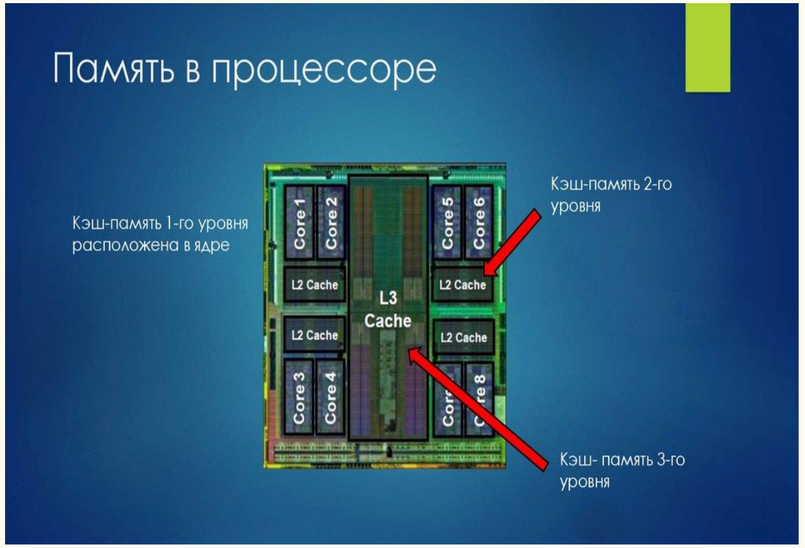


Нужно больше указателей на указатели

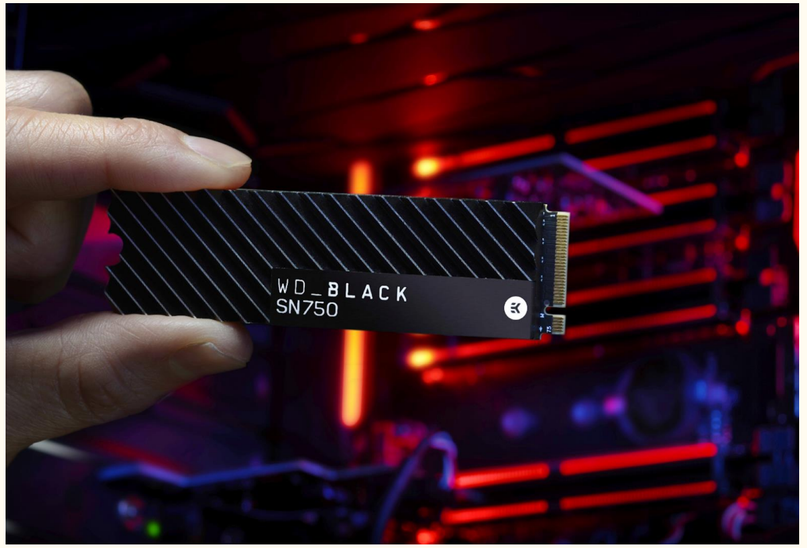
Нужно больше процессоров



Нужно больше ядер



Нужно больше SSD



С каким железом работали разработчики

1993

Видеокарта : VGA 512 кб
Жесткий диск : 100 мб
Звуковая карта: PC speaker
Процессор : 386SX 33Mhz
ОЗУ : 4 Мб

1996

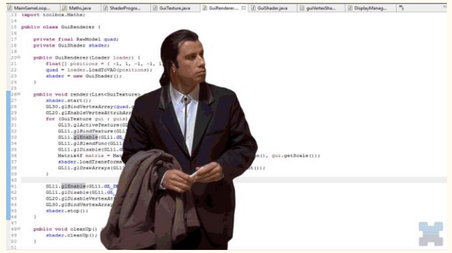
Видеокарта : Voodoo 1 4мб
Жесткий диск : 700 мб
Звуковая карта: Sound Blaster
Процессор : Pentium 166Mhz
ОЗУ : 16 Мб

1999

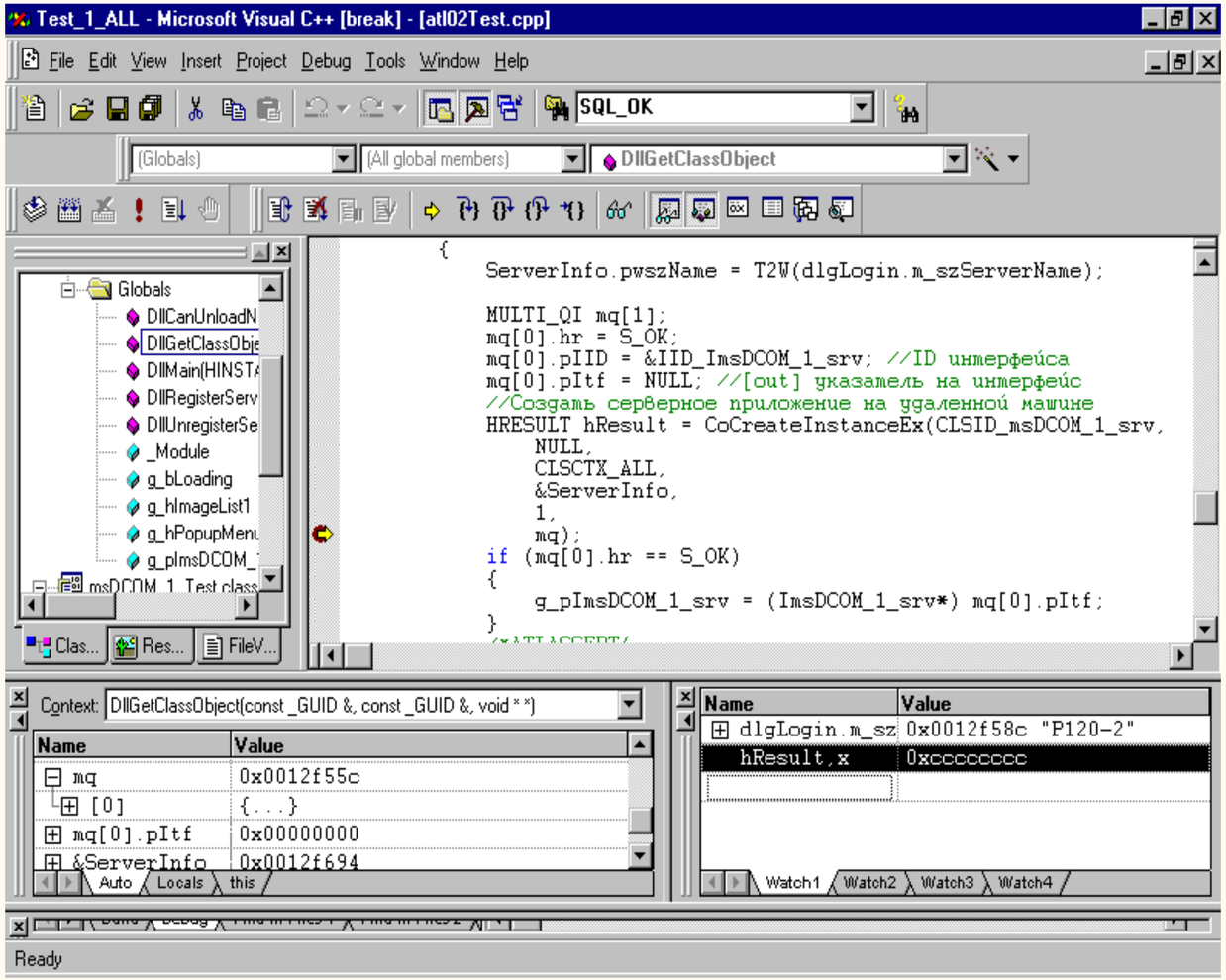
Видеокарта : Nvidia Riva TNT 16 мб
Жесткий диск : 2 гб
Звуковая карта: Sound Blaster
Процессор : Pentium 450 Mhz
ОЗУ : 64 Мб

Какие инструменты использовали

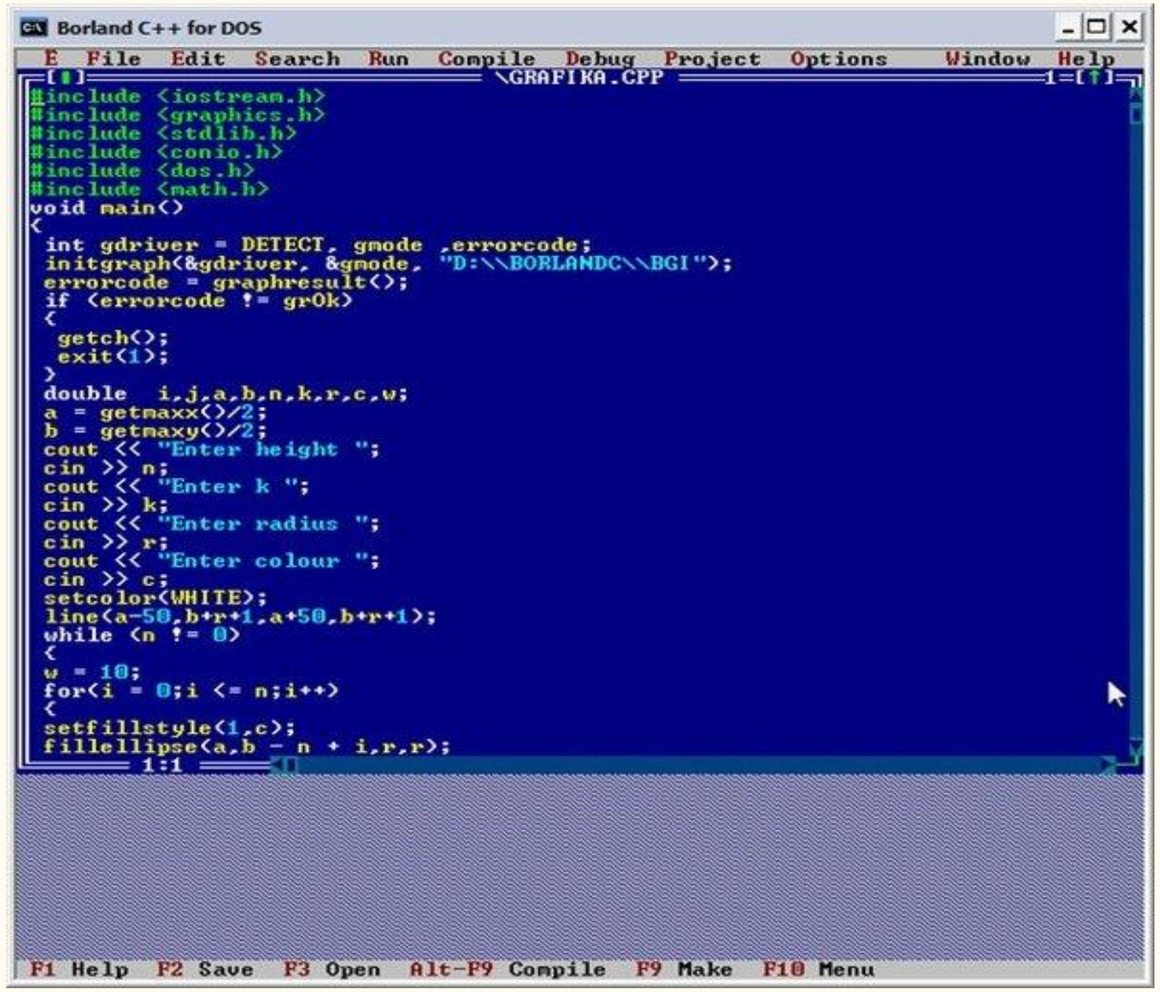
Автокомплит? А как перейти в класс? Когда скомпилится?



Visual C++ 6.0

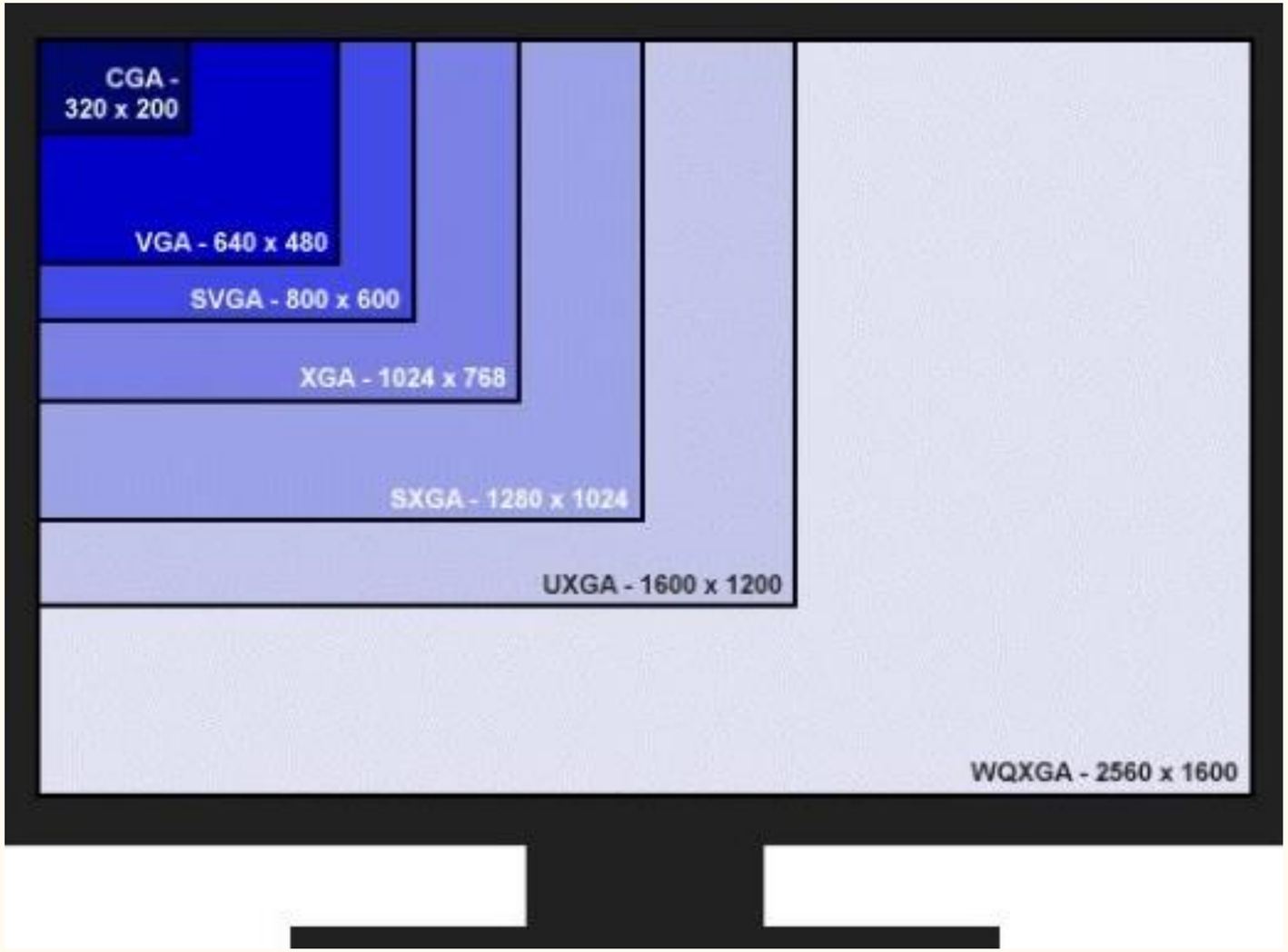


Borland C++ For Dos



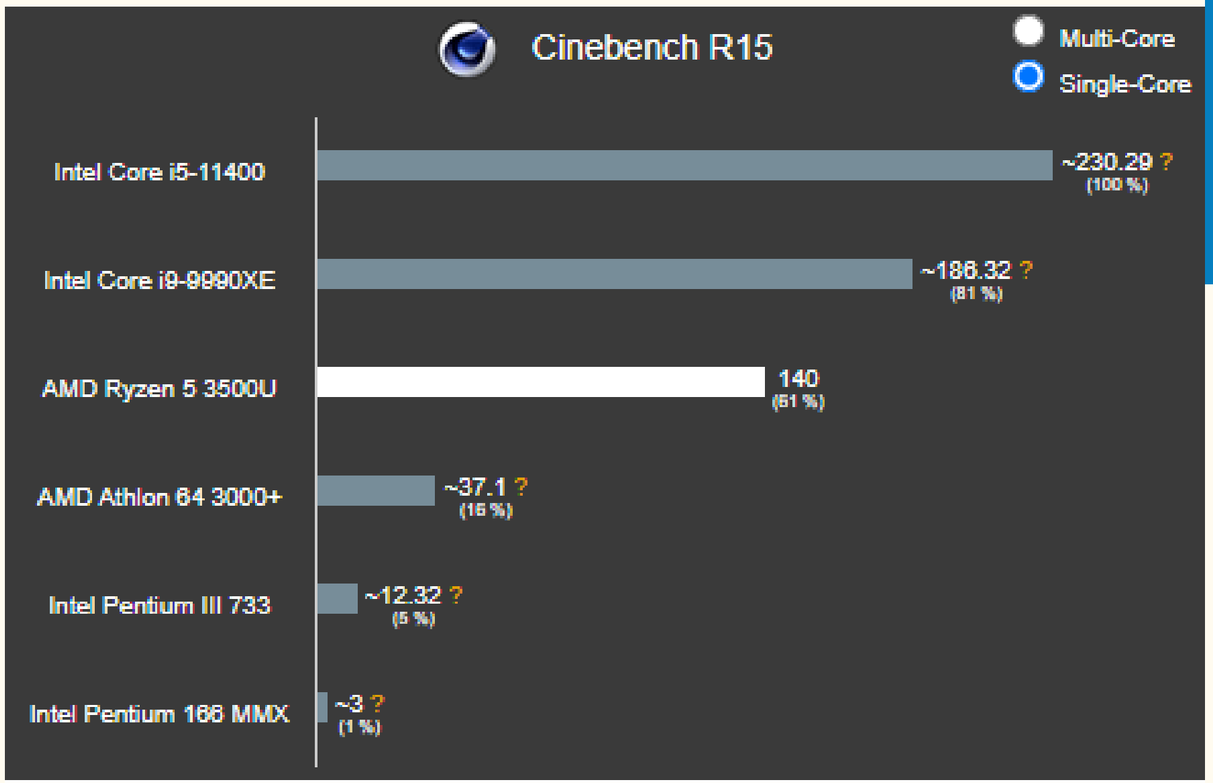
На сколько выросло разрешение экрана

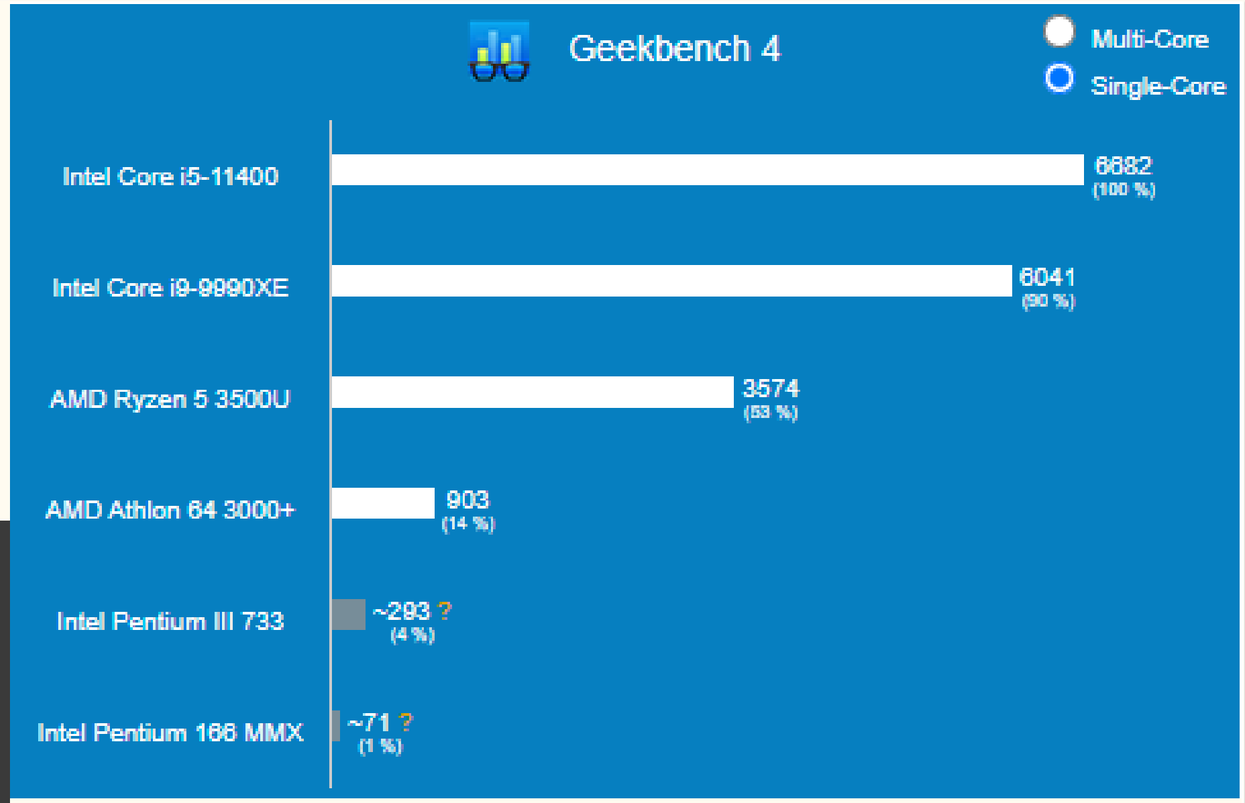
- Мы играли в разрешении 320x200
- Да это было в прошлом тысячелетии
- Разрешение экрана увеличилось на порядок
- Количество видимых пикселей увеличилось в 30 раз



Производительность тогда и сейчас, на одно ядро

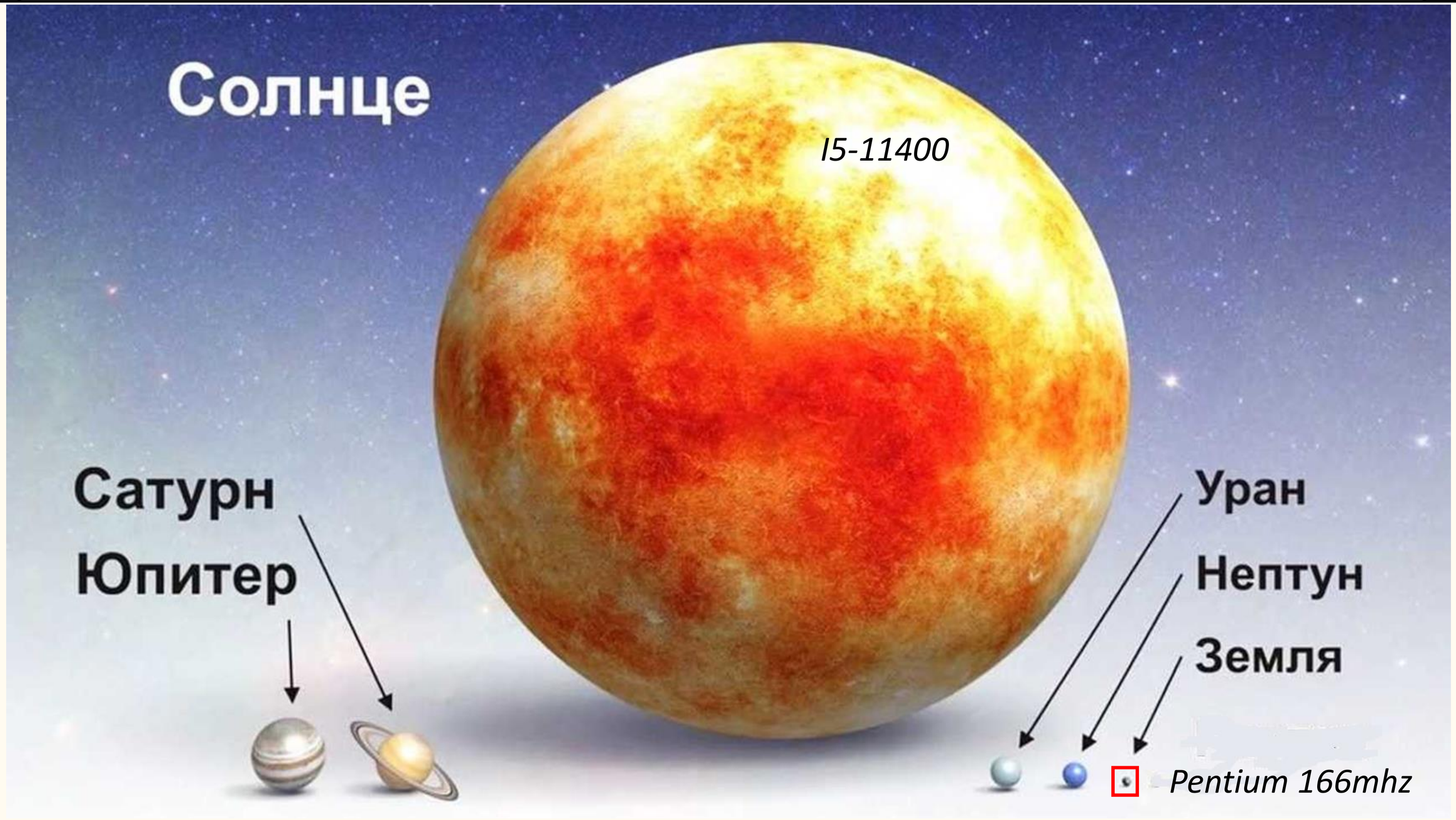
- Pentium 166 в сравнении с современными процессорами, даже не погрешность
- Скорость возросла в 200 раз





- Все также тормозит
- И мы также ждем загрузку ОС

Визуально различия выглядят примерно так



3D ускоряется

256 цветов, палитры и костыли

Шейдеры, все больше операций просчитывается на видеокарте

Появились аппаратные полигоны и наложение текстур

Гигабайты текстур, десятки миллионов треугольников в кадре

В какие игры играли

- Пиксельное месиво
- Низкое разрешение
- Шумный кулер
- Сложный геймплей

The Elder Scrolls | The Elder Scrolls II: Daggerfall (1996)

- Мир площадью 160 000 квадратных километров
- Наполненный 15 000 городов
- Население 750 000 неигровых персонажей
- Динамически подстраивающийся под игрока мир
    - Смена дня и ночи
    - Смена времен года
    - Открытый мир
    - 100500 квестов
    - Очень много багов

- Системные требования
- 486DX2 — 66 МГц,
- 8 Мб ОЗУ,
- 32 Мб HDD,
- Видеокарта на 32-х битной шине VLB/PCI,

Kingdom Rush Origins (2014)

Управляйте своей эльфийской армией и защищайте мистические земли от морских змей, злых волшебников и волнуйте за волной племени гноллов, все с помощью новых башен, героев и заклинаний, чтобы отбивать всех последних злодеев.

МИНИМАЛЬНЫЕ:
ОС: Windows 7
Процессор: Dual Core CPU
Оперативная память: 1 GB ОЗУ
Видеокарта: OpenGL 3.0 512MB
Место на диске: 500 MB

РЕКОМЕНДОВАННЫЕ:
Оперативная память: 2 GB ОЗУ
Видеокарта: OpenGL 3.0 1.0GB VRAM.

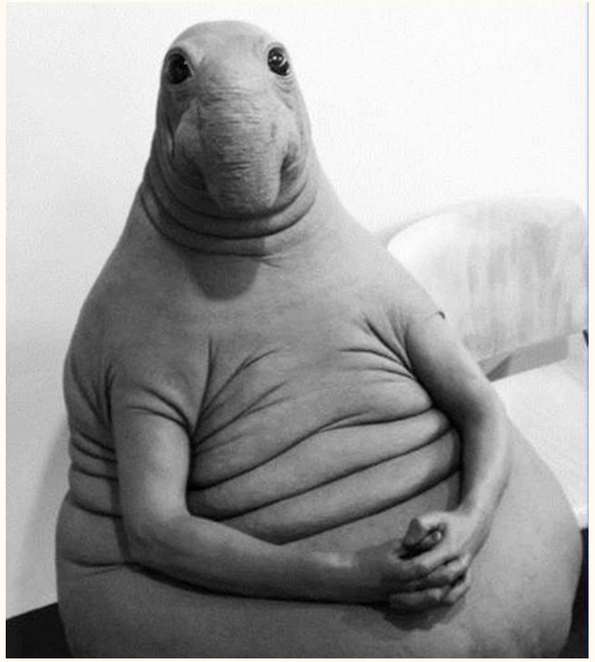
Ещё один пример из недавнего прошлого

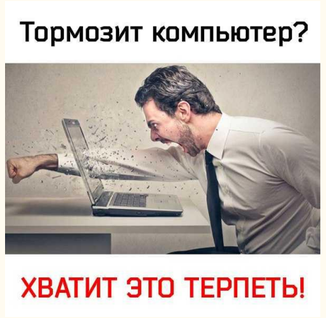
-Операционная система: Windows XP
-Процессор: Pentium 4;
-Оперативная память: 1 ГБ;
-Видеокарта: DirectX9-совместимая с 512 МБ видеопамяти;
-Звук: DirectX-совместимая звуковая карта;
-Жесткий диск: 6 ГБ;

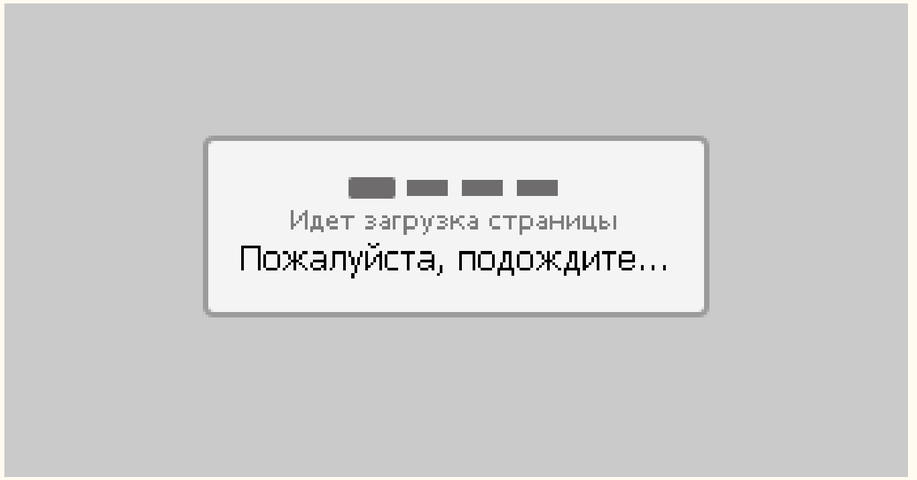
-Операционная система: Windows XP/Vista/7 (32 или 64 бита);
-Процессор: двухядерный с частотой не менее 2 ГГц;
-Оперативная память: 2 ГБ;
-Видеокарта: DirectX9-совместимая с 512 МБ видеопамяти;
-Звук: DirectX-совместимая звуковая карта;
-Жесткий диск: 6 ГБ;

Мы постоянно ждём!

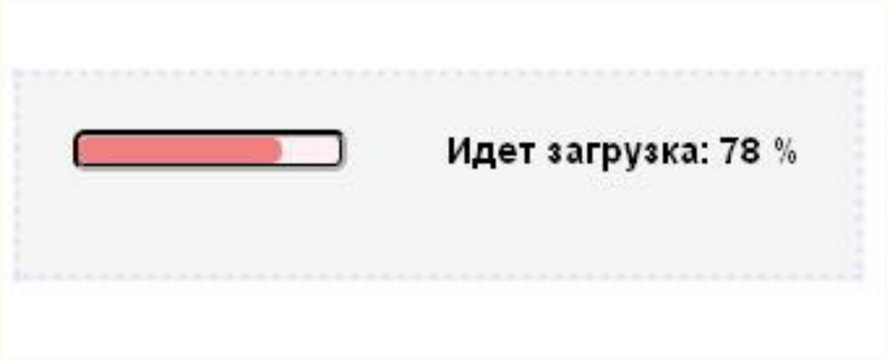
Ждун
Характер ожидающий







- **Мы ждём пока загрузится браузер**
- **Мы ждём пока загрузится сайт**
- **Мы ждем пока жирный JS фреймворк обработает нажатие**
- **Мы ждем когда заходим в настройки Windows**
- **Мы ждём, постоянно ждем пока процессор перемелет неоптимальный код**
- **Ждем загрузки ОС**
- **Мы ждем перемен!**



Решение мелких задач неподходящими инструментами

- Уже вышел просмотрщик фотографий на движке Unity?
- Огромные фреймворки для “вывода строки”
- Программирование стало неоправданно сложным
- Мы научились пользоваться инструментами, но забыли основы
- Мы забиваем гвозди комбайном
- Каждая программа рассчитывает возраст вселенной

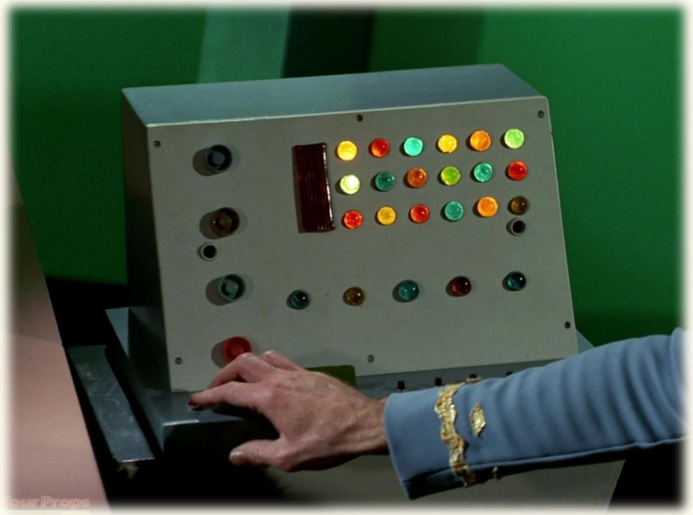
- ОЗУ конечно
- operator::new может быть долгим
- Помни о L1, L2, L3 кэшах
- Процессор только выполняет твой код

Есть что-то чарующее в старом железе

- Внешний вид
- Необычный дизайн
- Запоминающийся интерфейс

- Звуки включения и при работе
- Не было бездушного RGB

2015



- Дизайн был практичным
- Если что-то мигало это был индикатор системного процесса

2023

А что если написать свой фреймворк?

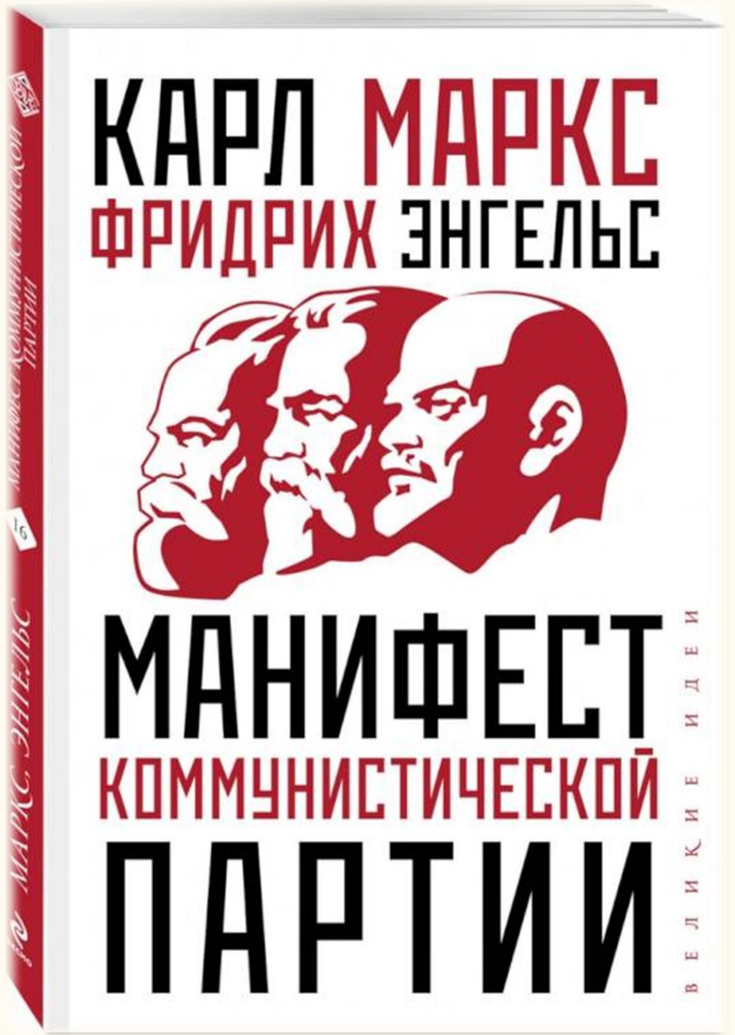
- Наличие инструментов разработки
- Системы изучены и задокументированы
- Исчерпывающая информация на тематических форумах
- Доступность старого железа за копейки (перепаять кондёр)

Основополагающие принципы разработки фреймворка

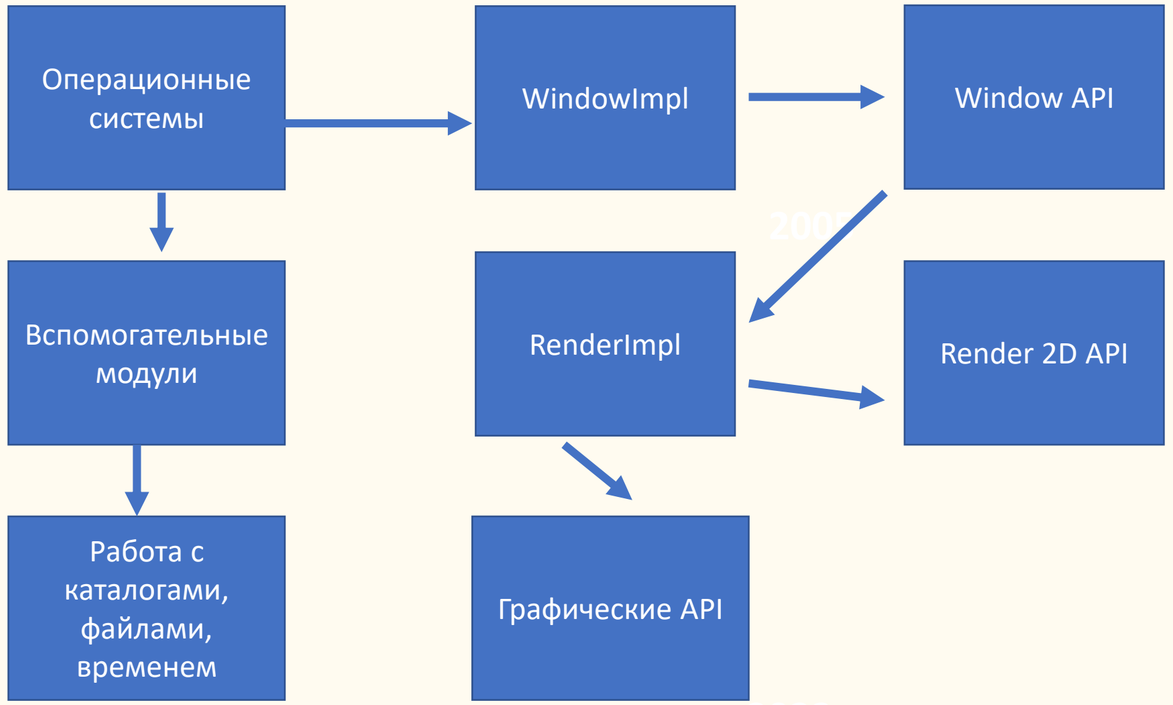
- Кроссплатформенность – обеспечение работы как на старых так и на новых системах.
- Поддержка всевозможных графических API OpenGL, Glide, Vulkan и DirectX начиная с 5.0 по 12.0
- Удобный высокоуровневый API.
- API одинаков для всех ОС и графических API
- Открытый исходный код по лицензии Boost Software License.
- Фреймворк совместим с C++ 98 и выше
- Производительность
- Запуск начиная с Windows 95
- Обеспечение сборки в Linux 2000-ых годов
- Совместимость со старым железом начиная с S3 Trio, Voodoo

2005

2023



Из чего состоит фреймворк



2015

Пора писать фреймворк для Windows 95

2023

Использование кастомных аллокаторов

- Реализация поддержки аллокаторов через стандартный интерфейс
- Добавление своих аллокаторов под задачу

```cpp
class Allocator
{
public:
    enum
    {
        Kb = 1024,
        Mb = Kb * 1024,
        Gb = Mb * 1024
    };
    virtual void* Allocate(size_t bytes) = 0;
    virtual void Deallocate(void* ptr) = 0;
    virtual size_t UsedBytes() = 0;
    virtual void Reset() = 0;
private:
};
```

- Использование аллокатора при загрузке изображений

```cpp
static LDL::Allocators::Allocator* StbImageAllocator;

#define STBI_MALLOC(sz)                         StbImageAllocator->Allocate(sz);
#define STBI_FREE(p)                            StbImageAllocator->Deallocate(p)

void* ReallocateSized(void* ptr, size_t Oldbytes, size_t Newbytes)
{
    void* result = NULL;

    if (!ptr)
    {
        result = StbImageAllocator->Allocate(Newbytes);
    }
    else
    {
        if (Oldbytes < Newbytes)
        {
            result = StbImageAllocator->Allocate(Newbytes);
            memcpy(result, ptr, Oldbytes);
        }
        else
        {
            result = ptr;
        }
    }

    return result;
}

#define STBI_REALLOC_SIZED(p,oldsz,newsz)  ReallocateSized(p, oldsz, newsz)
```

Пример загрузки изображения

- Создаем аллокатор и выделяем память

```
FixedLinear allocator(Allocator::Mb * 4);
ImageLoader loader(&allocator);
```

- Используя раннее созданный аллокатор, загружаем изображение
- Создаем текстуру и копируем данные изображения

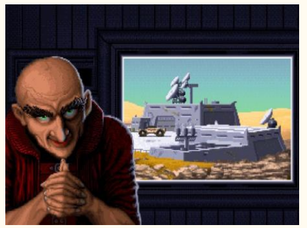
```
loader.Load("trehmachtovyiy-korabl-kartina-maslom-60x50_512x.jpg");
Texture image(&renderContext, loader.Size(), loader.Pixels(), loader.BytesPerPixel());
```

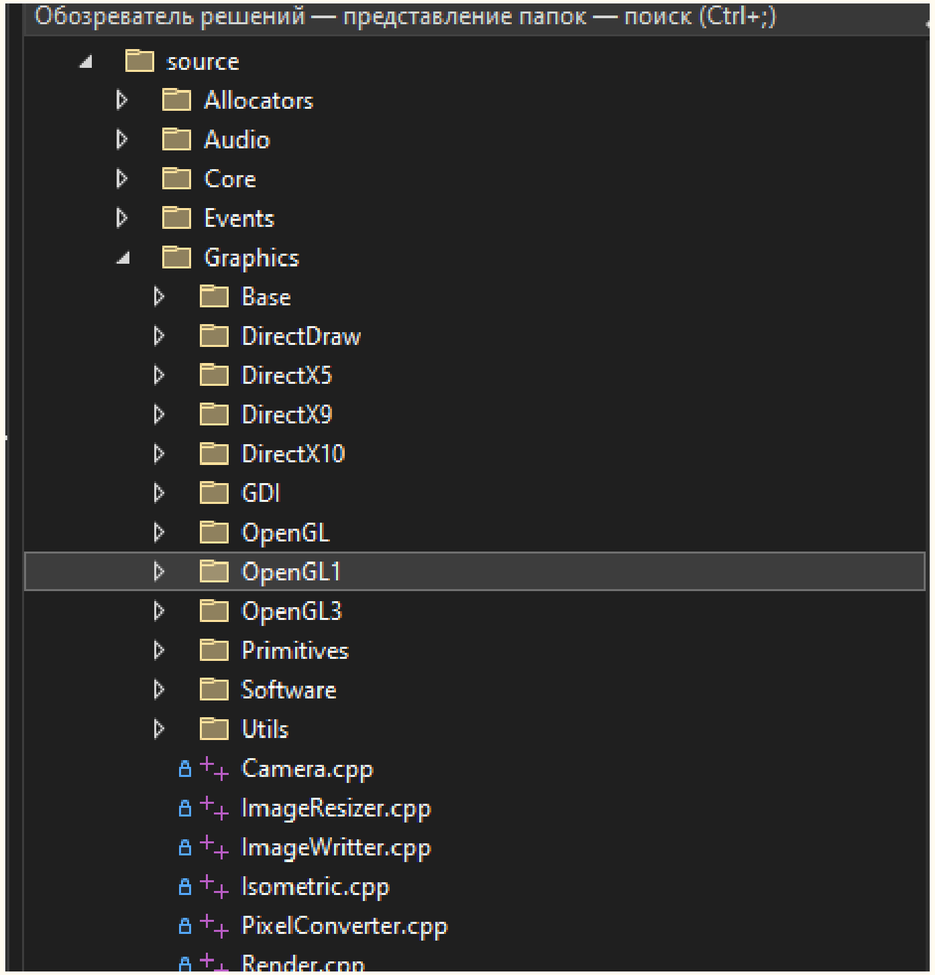
```
void ImageLoader::Load(const std::string& path)
{
    if (path.empty())
        throw LDL::Core::RuntimeError("Argument path is empty");

    Clear();

    int width         = 0;
    int height        = 0;
    int bytesPerPixel = 0;

    _Pixels = stbi_load(path.c_str(), &width, &height, &bytesPerPixel, STBI_default);

    if (width <= 0 || height <= 0 || bytesPerPixel <= 0 || _Pixels == NULL)
        throw LDL::Core::RuntimeError("stbi_load " + path + " failed");

    _Size = Point2u(width, height);
    _BytesPerPixel = bytesPerPixel;
}
```

- Очищаем аллокатор при загрузке изображения
- Перезаписываем новыми данными
- Отсутствие new при любом количестве загружаемых изображений

Pimpl всему голова

Pimpl — хорошо известная идиома C++, которая позволяет программисту отделить интерфейс класса от его реализации в такой степени, как того не позволяет сделать C++ напрямую. Положительными побочными эффектами использования d-указателя являются ускорение компиляции, упрощение реализации семантики транзакций и возможность сделать реализацию, потенциально более эффективную во времени выполнения, используя расширенные способы композиции.






```cpp
#include <LDL/Graphics/Render.hpp>

#ifdef LDL_RENDER_SOFTWARE
#include "Software/RenderImpl.hpp"
#elif LDL_RENDER_GDI
#include "GDI/RenderImpl.hpp"
#elif LDL_RENDER_OPENGL1
#include "OpenGL1/RenderImpl.hpp"
#elif LDL_RENDER_OPENGL3
#include "OpenGL3/RenderImpl.hpp"
#elif LDL_RENDER_DIRECTX1
#include "DirectDraw/RenderImpl.hpp"
#elif LDL_RENDER_DIRECTX5
#include "DirectX5/Direct3D/RenderImpl.hpp"
#elif LDL_RENDER_DIRECTX9
#include "DirectX9/Direct3D/RenderImpl.hpp"
#elif LDL_RENDER_DIRECTX10
#include "DirectX10/Direct3D/RenderImpl.hpp"
#endif

using namespace LDL::Graphics;

Render::Render(RenderContext* renderContext, Window* window) :
    _RenderImpl(new RenderImpl(renderContext->GetRenderContextImpl(), window))
{
}

Render::~Render()
{
    delete _RenderImpl;
}
```

Но он не простой, а ~~золотой~~ быстрый!

- Память для классов выделяется из фиксированных pool’ов
- При деструкторе память возвращается в pool
- O(1) для выделения памяти под объект

```hpp
#ifndef LDL_Core_FastPimpl_hpp
#define LDL_Core_FastPimpl_hpp

#include <LDL/Config.hpp>
#include <LDL/Core/Types.hpp>

namespace LDL
{
    namespace Core
    {
        class LDL_EXPORT FastPimpl
        {
        public:
            void* operator new(size_t bytes);
            void  operator delete(void* ptr);
        private:
        };
    }
}

#endif
```

```hpp
#ifndef LDL_Graphics_Texture_hpp
#define LDL_Graphics_Texture_hpp

#include <LDL/Core/FastPimpl.hpp>
#include <LDL/Graphics/Primitives/Point2u.hpp>
#include <LDL/Graphics/RenderContext.hpp>

namespace LDL
{
    namespace Graphics
    {
        class TextureImpl;

        class LDL_EXPORT Texture : public LDL::Core::FastPimpl
        {
        public:
            Texture(RenderContext* renderContext, const Point2u& size, uint8_t* pixels, uint8_t bytesPerPixel);
            ~Texture();
            const Point2u& Size();
            TextureImpl* GetTextureImpl();
        private:
            TextureImpl* _TextureImpl;
        };
    }
}

#endif
```

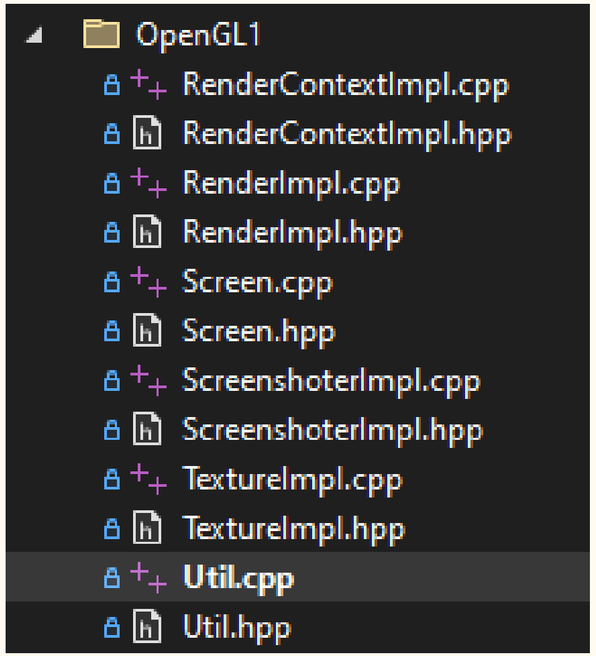
2023

Фреймворк для Windows 95

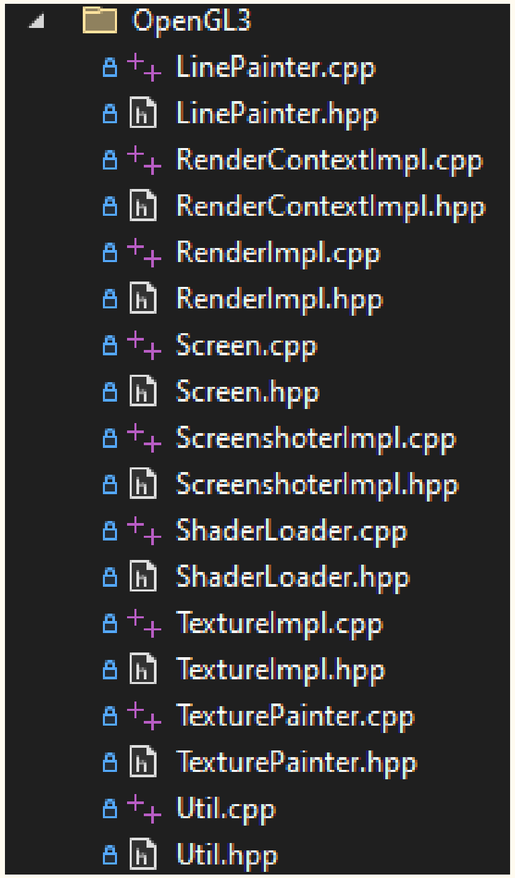
Трудозатраты по поддержке старых систем

Поддержка API для 2D графики

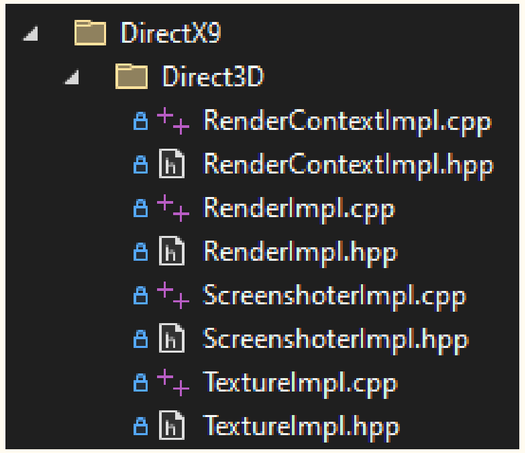
OpenGL 1.2
- 6 файлов реализации
- 1000 строк кода



OpenGL 3.0
- 9 файлов реализации
- 900 строк кода

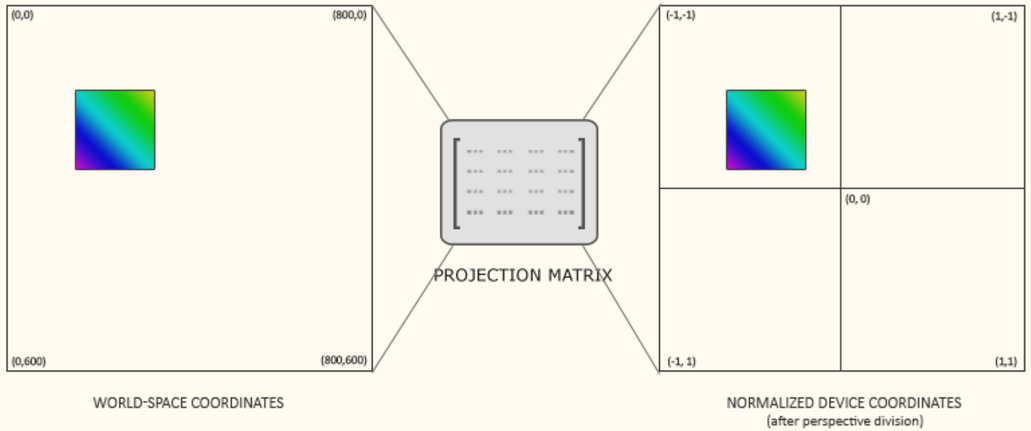


Direct3D 9.0
- 4 файла реализации
- 500 строк кода



- Поддержка для каждой платформы 3000 строк кода. На фоне фреймворка это копейки.
- Возрастает когнитивная нагрузка
- В любом случае затрачивается больше времени

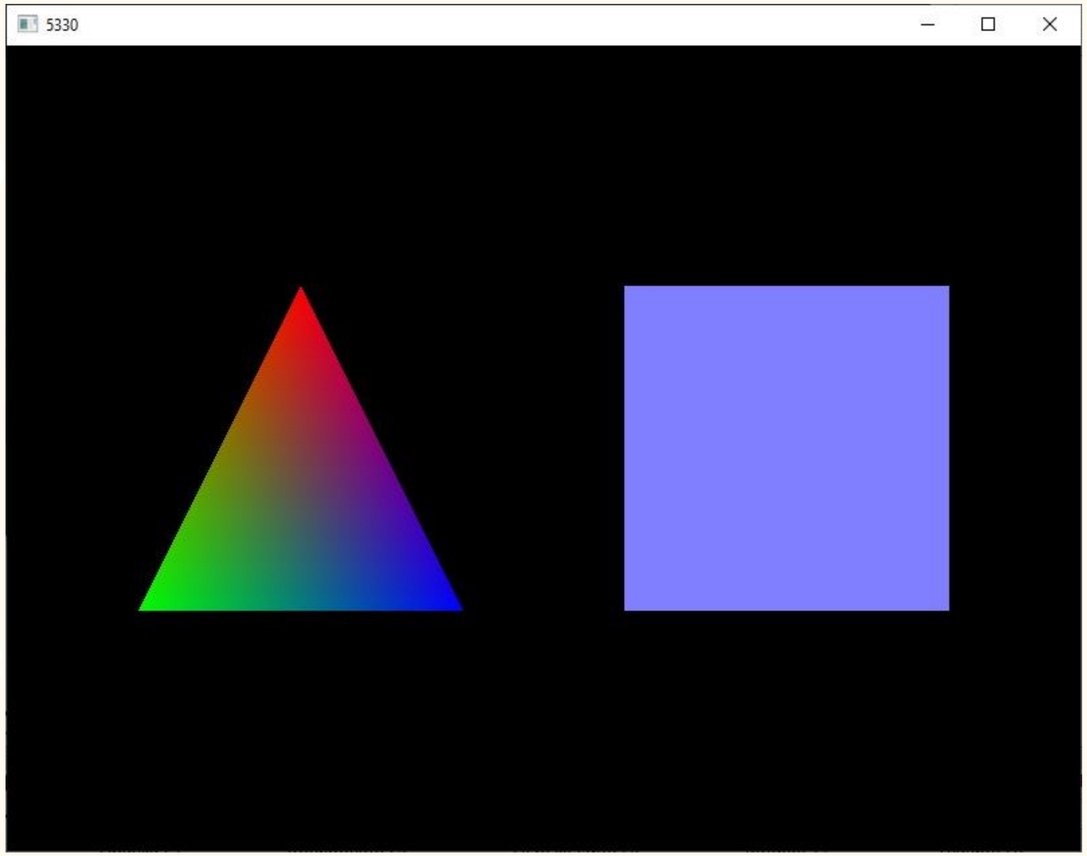
Как работает 2D в 3D



- Используя графическое API, задаем ортографическая проекцию
- Указываем размеры области рисования

- После установки проекции, получаем привычные нам 2D координаты

```
void RenderImpl::Begin()
{
    GL_CHECK(glViewport(0, 0, (GLsizei)_Window->Size().PosX(), (GLsizei)_Window->Size().PosY()));

    Mat4f projection;
    Mat4f modelView;

    GL_CHECK(glMatrixMode(GL_PROJECTION));
    projection = Ortho(0.0f, (float)_Window->Size().PosX(), (float)_Window->Size().PosY(), 0.0f, 0.0f, 1.0f);
    GL_CHECK(glLoadMatrixf(projection.Values()));

    GL_CHECK(glMatrixMode(GL_MODELVIEW));
    GL_CHECK(glLoadMatrixf(modelView.Values()));
}
```



Преодолевая ограничения старых видеокарт ч.1

Видеокарта Voodoo 2 (1998):

- **Максимальное разрешение текстуры 256x256**
- **8 мб видеопамяти (кеш в современном CPU)**
- **Максимальное разрешение в 3D 1024x768**

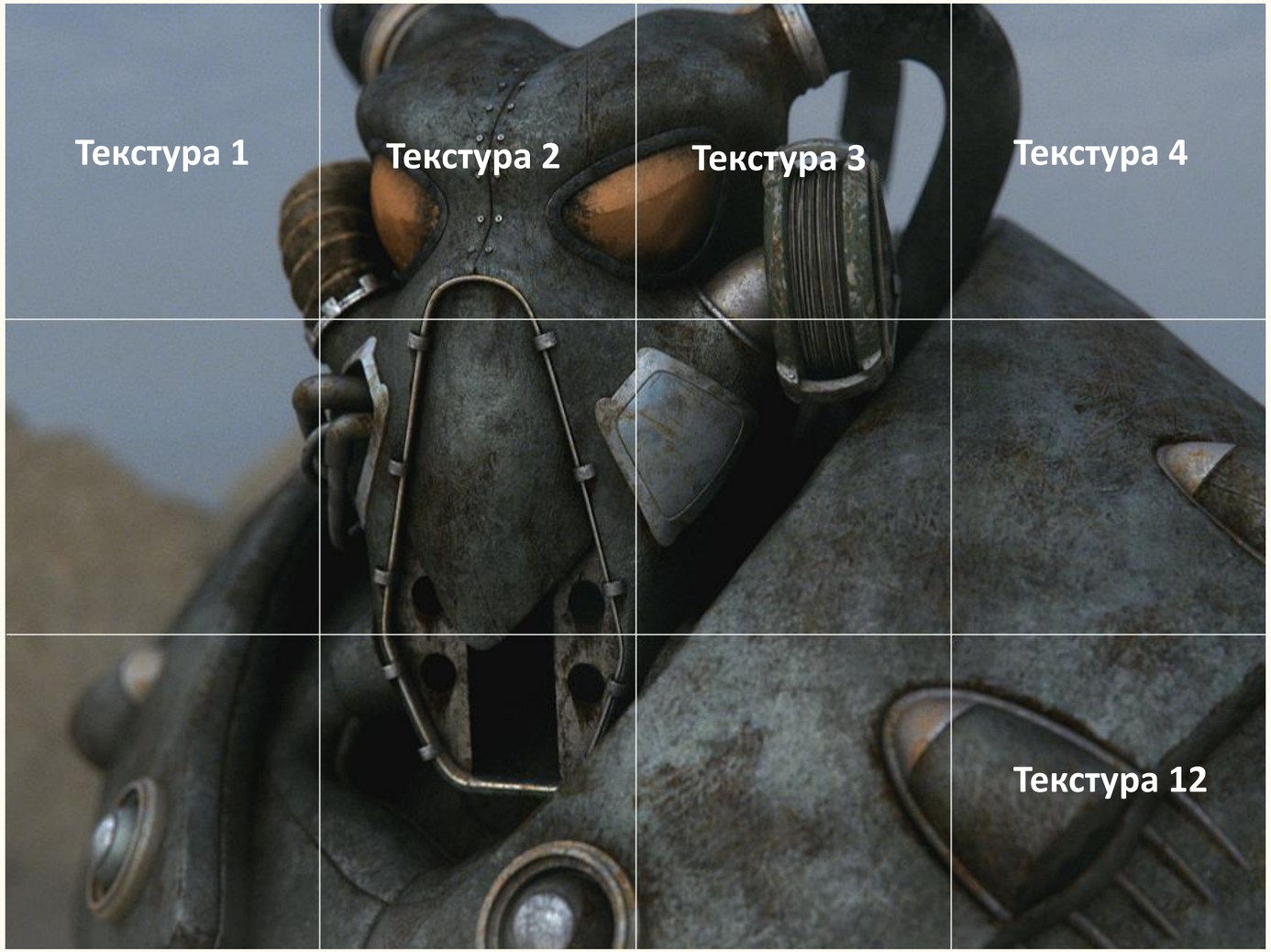
Нужно вывести картинку с разрешением 1024x768 с аппаратным ускорением

- **Разрешение картинки уменьшено в 4 раза**
- **Потеря информации составила 75%**

Преодолевая ограничения старых видеокарт ч.2

- Потеря пикселей
- Появились лесенки
- Выглядит аутентично:)

Преодолевая ограничения старых видеокарт ч.3



- Создаем класс TextureCutter
- Данный класс при загрузке картинки, нарезает на количество текстур в зависимости от разрешения самого растрового файла

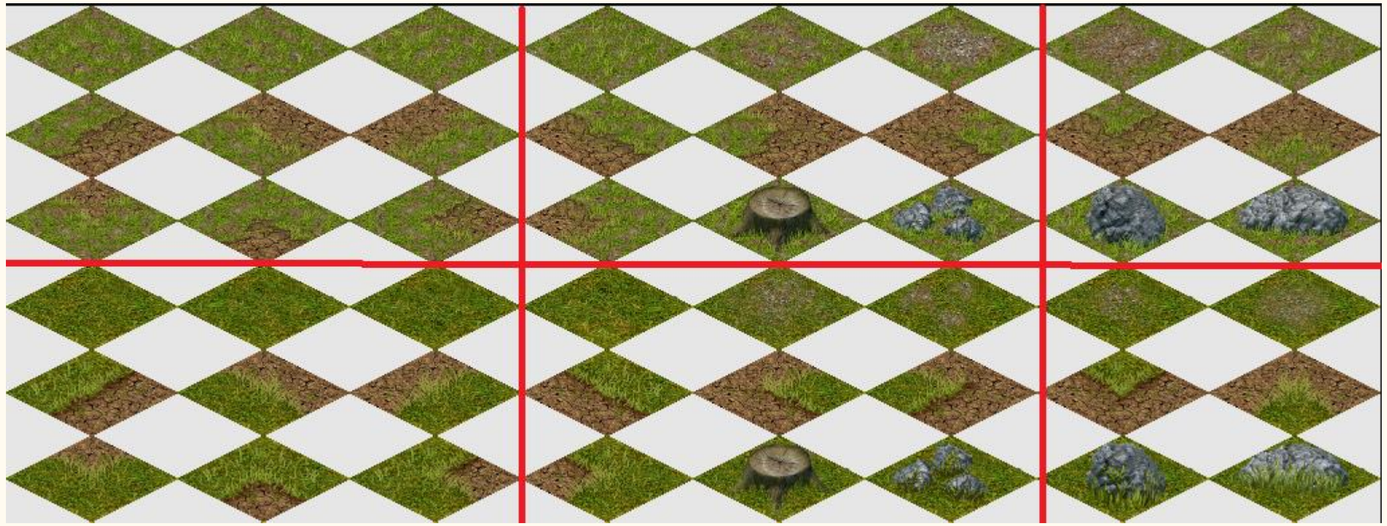
- Каждый квадрат это текстура
- 4x3 = 12 текстур на файл
- Минимум 12 вызовов отрисовки
- Более сложный алгоритм
- Требуется дополнительный код для правильного масштабирования и вывода части изображения
- Сложность скрыта под стандартным API
- Получаем аппаратную поддержку масштабирования и рисования

```
void Draw(Texture* image, const Point2u& pos, const Point2u& size);
void Draw(Texture* image, const Point2u& pos);
void Draw(Texture* image, const Point2u& dstPos, const Point2u& srcPos, const Point2u& srcSize);
void Draw(Surface* image, const Point2u& pos, const Point2u& size);
void Draw(Surface* image, const Point2u& pos);
```

Преодолевая ограничения старых видеокарт ч.4

- **Вывод картинок с высоким разрешением**
- **Вывод интерфейсов игр**

Преодолевая ограничения старых видеокарт ч.5



- Создание текстурных атласов налет
- Отправка на отрисовку отсортированные вершины

- Для пользователя фреймворка, код остается линейным
- Внутренняя структура сопоставляет, имя текстуры и набор вершин группирует в массивы, для уменьшения вызовов отрисовки

2005 2015

```
for (size_t rows = 0; rows < mapSize.PosX(); rows++)
{
    for (size_t cols = 0; cols < mapSize.PosY(); cols++)
    {
        size_t x = cols * tileSize.PosX() / 2;
        size_t y = rows * tileSize.PosY();

        Point2u pt = isometric.CartesianToIsometric(Point2u(x, y));

        size_t tx = tileSize.PosX() * tilesX[j];
        size_t ty = tileSize.PosY() * tilesY[j];
        j++;

        render.Draw(&image, Point2u(start.PosX() + pt.PosX() + dx, start.PosY() + pt.PosY() + dy), Point2u(tx, ty), tileSize);
    }
}
```

- Все эти сложности требуется решить, для обеспечения аппаратного ускорения графики
- Мы получаем аппаратное ускорение трансформаций и масштабирования

- В средней игре немало других объектов:
- Декорации
- Стены
- Игровые объекты

Все свое ношу с собой

- Зависимость от базовых библиотек на ОС
- Для Windows базовый WinAPI
- Для Linux Xlib
- Остальные библиотеки лежат в исходниках
- stb_image
- stb_resize
- stb_vorbis
- dr_libs для работы с форматами audio
- Header only library или маленькие портативные библиотеки
- Пришлось написать свой функционал по работе с матрицами и векторами

2005

2015

2023

Пора писать фреймворк для Windows 95

```cpp
bool Window::GetEvent(LDL::Events::Event& event)
{
    return _WindowImpl->GetEvent(event);
}

bool Window::WaitEvent(LDL::Events::Event& event)
{
    return _WindowImpl->WaitEvent(event);
}

void Window::StopEvent()
{
    _WindowImpl->StopEvent();
}

void Window::Title(const std::string& title)
{
    _WindowImpl->Title(title);
}
```

Главный класс отвечающий за взаимодействие со окном

```cpp
class MainWindow { ... };
```

Реализация окна с учетом графического API
WindowImpl включает MainWindow

```cpp
class WindowImpl { ... };
```

Класс который виден наружу программисту

```cpp
class LDL_EXPORT Window { ... };
```

2015

2023

Пора писать фреймворк для Windows 95

Не забывай о братьях меньших

- Поверх C++ реализован C API для удобства создания биндингов
- В будущем планирую автоматизировать процесс биндинга для разных языков

```c
#ifndef LDL_Render_h
#define LDL_Render_h

#include <LDLC/LDL_Export.h>
#include <LDLC/LDL_RenderContext.h>
#include <LDLC/LDL_Window.h>

#ifdef __cplusplus
extern "C" {
#endif

typedef struct LDL_Texture LDL_Texture;
typedef struct LDL_Render LDL_Render;

LDL_EXPORT LDL_Render* LDL_RenderNew(LDL_RenderContext* renderContext, LDL_Window * window);
LDL_EXPORT void LDL_RenderFree(LDL_Render* render);

LDL_EXPORT void LDL_RenderBegin(LDL_Render* render);
LDL_EXPORT void LDL_RenderEnd(LDL_Render* render);

LDL_EXPORT size_t LDL_RenderGetSizeX(LDL_Render* render);
LDL_EXPORT size_t LDL_RenderGetSizeY(LDL_Render* render);

LDL_EXPORT void LDL_RenderClear(LDL_Render* render);
LDL_EXPORT void LDL_RenderSetColor(LDL_Render* render, uint8_t r, uint8_t g, uint8_t b, uint8_t a);
LDL_EXPORT void LDL_RenderFill(LDL_Render* render, size_t x, size_t y, size_t w, size_t h);
LDL_EXPORT void LDL_RenderLine(LDL_Render* render, size_t x, size_t y, size_t w, size_t h);

LDL_EXPORT void LDL_RenderDrawTexture1(LDL_Render* render, LDL_Texture* texture, size_t x, size_t y);
LDL_EXPORT void LDL_RenderDrawTexture2(LDL_Render* render, LDL_Texture* texture, size_t x, size_t y, size_t w, size_t h);
LDL_EXPORT void LDL_RenderDrawTexture3(LDL_Render* render, LDL_Texture* texture, size_t dx, size_t dy, size_t x, size_t y, size_t w, size_t h);

#ifdef __cplusplus
}
#endif
```

Ограничений нет

- Фреймворк не накладывает ограничений на использование графического API
- Используйте фреймворк как тонкую прослойку для вашего проекта

```
float vertices[] = {
    // positions          // colors           // texture coords
     0.5f,  0.5f, 0.0f,   1.0f, 0.0f, 0.0f,   1.0f, 1.0f, // top right
     0.5f, -0.5f, 0.0f,   0.0f, 1.0f, 0.0f,   1.0f, 0.0f, // bottom right
    -0.5f, -0.5f, 0.0f,   0.0f, 0.0f, 1.0f,   0.0f, 0.0f, // bottom left
    -0.5f,  0.5f, 0.0f,   1.0f, 1.0f, 0.0f,   0.0f, 1.0f  // top left
};
unsigned int indices[] = {
    0, 1, 3, // first triangle
    1, 2, 3  // second triangle
};
unsigned int VBO, VAO, EBO;
glGenVertexArrays(1, &VAO);
glGenBuffers(1, &VBO);
glGenBuffers(1, &EBO);

glBindVertexArray(VAO);

glBindBuffer(GL_ARRAY_BUFFER, VBO);
glBufferData(GL_ARRAY_BUFFER, sizeof(vertices), vertices, GL_STATIC_DRAW);

glBindBuffer(GL_ELEMENT_ARRAY_BUFFER, EBO);
glBufferData(GL_ELEMENT_ARRAY_BUFFER, sizeof(indices), indices, GL_STATIC_DRAW);

glVertexAttribPointer(0, 3, GL_FLOAT, GL_FALSE, 8 * sizeof(float), (void*)0);
glEnableVertexAttribArray(0);
glVertexAttribPointer(1, 3, GL_FLOAT, GL_FALSE, 8 * sizeof(float), (void*)(3 * sizeof(float)));
glEnableVertexAttribArray(1);
glVertexAttribPointer(2, 2, GL_FLOAT, GL_FALSE, 8 * sizeof(float), (void*)(6 * sizeof(float)));
glEnableVertexAttribArray(2);

GLuint texture = LoadTexture("resources/textures/container.jpg", loader);

while (window.GetEvent(report))
{
    render.Begin();
    glClearColor(0.2f, 0.3f, 0.3f, 1.0f);
    glClear(GL_COLOR_BUFFER_BIT);
    glBindTexture(GL_TEXTURE_2D, texture);
    ourShader.use();
    glBindVertexArray(VAO);
    glDrawElements(GL_TRIANGLES, 6, GL_UNSIGNED_INT, 0);

    render.End();

    if (report.Type == IsResize)
    {
        glViewport(0, 0, (GLsizei)report.Resize.Width, (GLsizei)report.Resize.Height);
    }

    if (report.Type == IsQuit || report.IsKeyPresed(KeyboardKey::Escape))
    {
        window.StopEvent();
    }
}
glDeleteVertexArrays(1, &VAO);
glDeleteBuffers(1, &VBO);
glDeleteBuffers(1, &EBO);
```

2005

2023

```
int main()
{
    try
    {
        Window window(Point2u(0, 0), Point2u(800, 600), "D3D Tutorial");

        RenderContext renderContext;
        Render render(&renderContext, &window);

        Direct3D9* direct3D9 = (Direct3D9*)renderContext.Context();
        g_pD3D = direct3D9->Direct;
        g_pd3dDevice = direct3D9->Device;

        LDL::Events::Event report;

        // Initialize Direct3D
        if (SUCCEEDED(InitD3D()))
        {
            // Create the geometry
            if (SUCCEEDED(InitGeometry()))
            {
                while (window.GetEvent(report))
                {
                    Rendering();

                    if (report.Type == LDL::Events::IsQuit)
                    {
                        window.StopEvent();
                    }
                }
            }
        }

        Cleanup();
    }
    catch (const LDL::Core::RuntimeError& error)
    {
        std::cout << error.what() << '\n';
    }

    return 0;
}
```

Сравнение и удобство

- Высокоуровневый ООП API
- Объявление классов не требует new
- Не требуется следить за временем жизни объектов
- Сложность максимальна скрыта за API с учетом производительности
- Модульный дизайн
- Наличие интеграционных тестов
- Разработчик может использовать фреймворк с любым стандартом языка C++

2005

2015

2023

Пора писать фреймворк для Windows 95

Какие инструменты использую при разработке

Windows 10
- **Visual Studio 2022**

Lubuntu 22.04.2 LTS
- **Visual Studio Code**
- **Git**
- **CMake**

2005

2015

Для старых систем тестирую под X86Box
Использую VirtualBox и Qemu

2023

Пора писать фреймворк для Windows 95

Для чего применять фреймворк

- Портирование старых игр
- Или разработка новых игр
- Легкая портабельность на разные ОС
- Написал один раз, скомпилится на поддерживаемых платформах

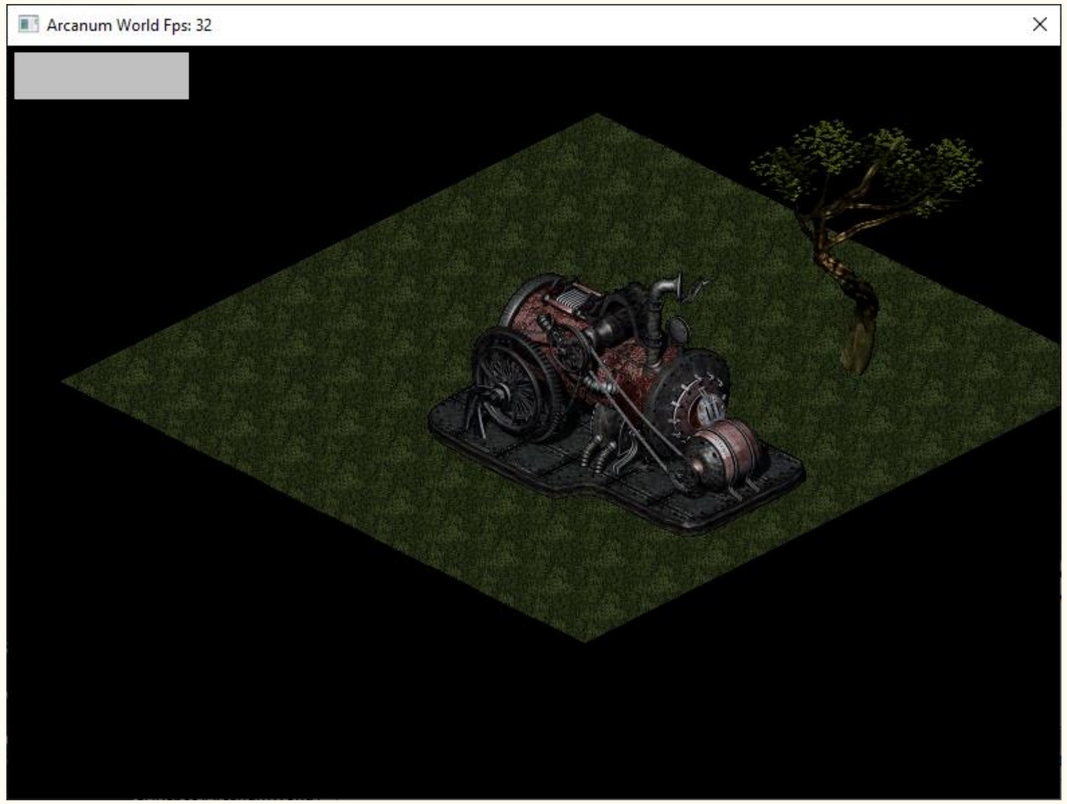
2005

2015

2023

Пора писать фреймворк для Windows 95

Проект для тестирования фреймворка - ArcanumWorld



- Проект по разработке движка для игры Arcanum
- Использую свой фреймворк
- Тестирую производительность
- Тестирую и проверяю идеи функционала
- Мне нравится делать 2D движок

https://github.com/JordanCpp/ArcanumWorld

Планы на будущее

- **Портирование под Android, Ms-Dos, Linux**
- **Обеспечение встроенной поддержки юникод**
- **Обеспечение потокобезопасности**
- **Увеличение числа поддерживаемых платформ**
- **Внедрить STL в проект для старых компиляторов**

Пора писать фреймворк для Windows 95

2023

https://github.com/JordanCpp/Lib-LDL

Итоги

- Разобрался как работают мультимедийные фреймворки
- Закрыл гештальт любопытства
- Узнал, что производительность не пустой звук
- Поработал со старым железом
- C++ 98 не так, что бы и устарел
- Написал много велосипедов

2005

2015

Пора писать фреймворк для Windows 95

2023

Благодарности

Хочу поблагодарить пользователей с сайтов:

www.GameDev.ru

www.OldGames.ru

Буду рад ответить на вопросы!

2005

2015

Пора писать фреймворк для Windows 95

2023