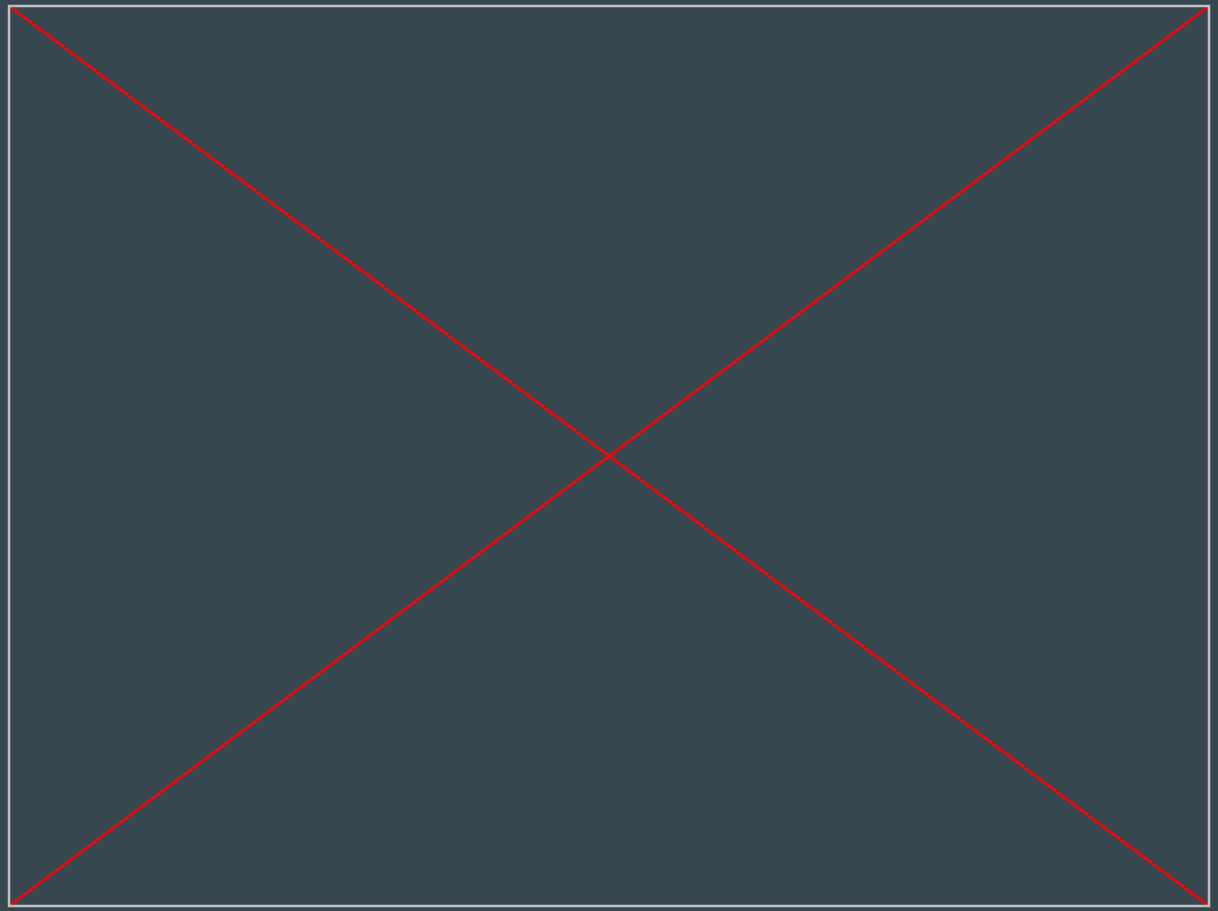
Geometry Rush

Doğa Sarp Sezer
22410004

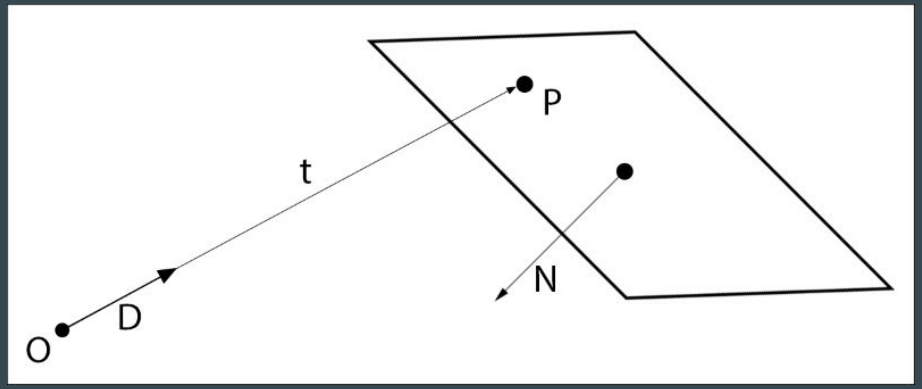


Debug Sistemi

Çarpışmalar

```
inline bool CircleCollision(Vector3& p1, float& r1, Vector3& p2, float& r2)
{
    //sqr length is much faster
    float pointDistance = (p1 - p2).SqrLength();
    float minDistanceForCollision = r1 + r2;

    return pointDistance < minDistanceForCollision * minDistanceForCollision;
}
```

Işınlar ve Düzlemler

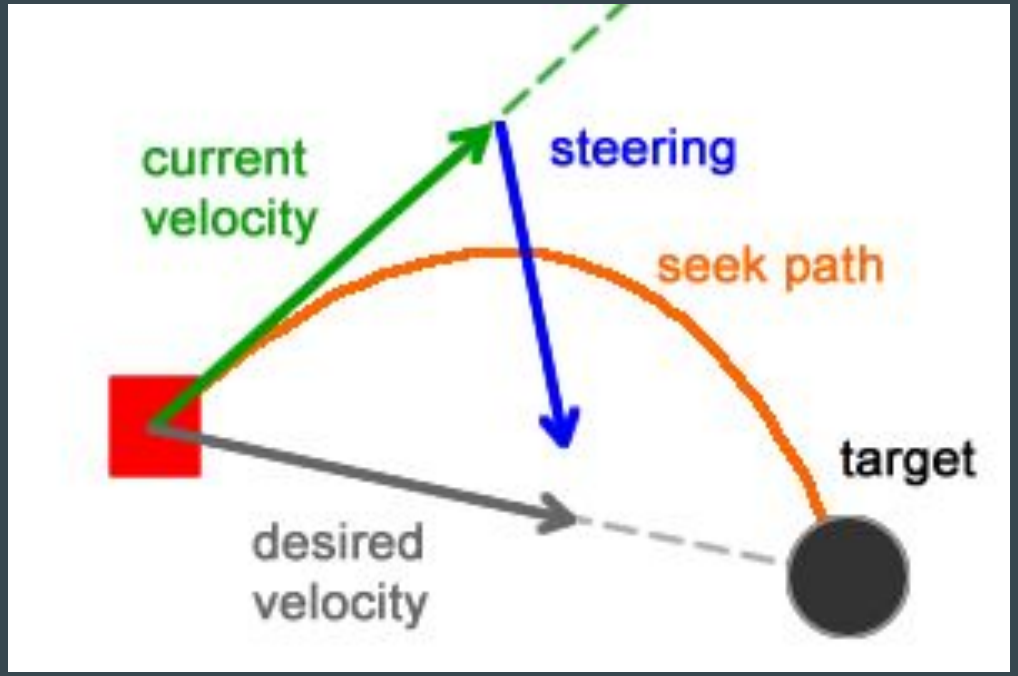



```
inline bool RaycastPlane(const Vector3& rayOrigin, const Vector3& rayDir,
    const Vector3& planePoint, const Vector3& planeNormal,
    Vector3& outHitPoint)
{
    float denom = planeNormal.Dot(rayDir);
    if (fabs(denom) < 0.0001f)
        return false;  // Ray is parallel to the plane, since the normal is always 90 to the plane

    float t = (planePoint - rayOrigin).Dot(planeNormal) / denom;
    if (t < 0)
        return false;  // Intersection is behind the ray origin

    outHitPoint = rayOrigin + rayDir * t;
    return true;
}
```

Zaman

```cpp
CustomTime& CustomTime::Instance()
{
    static CustomTime instance; // only created once
    return instance;
}

void CustomTime::Init(std::chrono::steady_clock::time_point startFrame)
{
    lastFrame = startFrame;
}

float CustomTime::Update()
{
    auto now = std::chrono::steady_clock::now();
    std::chrono::duration<float> elapsed = now - lastFrame;
    deltaTime = elapsed.count();
    lastFrame = now;
    return deltaTime;
}
```

Direksiyon Davranışları

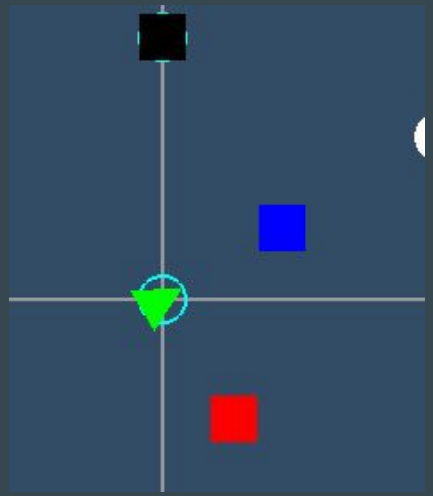


Bileşenler - Transform

```cpp
Vector3 Transform::Right()
{
    return Vector3(-matrix[0], -matrix[1], -matrix[2]).Normalized();
}

void Transform::UpdateMatrix()
{
    glMatrixMode(GL_MODELVIEW);
    glPushMatrix();
    glLoadIdentity();

    glTranslatef(position.x, position.y, position.z);
    glRotatef(rotation.z, 0, 0, 1);
    glRotatef(rotation.y, 0, 1, 0);
    glRotatef(rotation.x, 1, 0, 0);
    glScalef(scale.x, scale.y, scale.z);

    glGetFloatv(GL_MODELVIEW_MATRIX, matrix);
    glPopMatrix();
}

GLfloat* Transform::ToGLMatrix()
{
    return matrix;
}
```

```cpp
Transform::Transform(Vector3 pos, Vector3 rot, Vector3 scl)
    : position(pos), rotation(rot), scale(scl)
{
    UpdateMatrix();
}

void Transform::SetPosition(Vector3 newPos) { position = newPos; UpdateMatrix(); }
void Transform::SetRotation(Vector3 newRot) { r                  (); }
void Transform::SetScale(Vector3 newScale) { sc                  ; }

void Transform::Move(Vector3 movement) { position = position + movement; UpdateMatrix(); }
void Transform::RotateOnY(float angle)
{
    rotation.y += angle;
    UpdateMatrix();
}

Vector3 Transform::Forward()
{
    float pitch = rotation.x * 3.14159f / 180.0f;
    float yaw = rotation.y * 3.14159f / 180.0f;

    Vector3 forward;
    forward.x = cos(pitch) * sin(yaw);
    forward.y = sin(pitch);
    forward.z = cos(pitch) * cos(yaw);

    return forward;
}
```

Bileşenler - Collider



```cpp
class Collider
{
public:
    Collider(float radius) : radius(radius) {}

    float GetRadius(float scale)
    {
        return radius * scale;
    }

    void DrawDebug(Vector3 center, Vector3 scale)
    {
        SimpleCharacter* colliderDebug = new SimpleCharacter(SimpleGeo(CIRCLE_BOUNDS, cyan, 1.0f));
        colliderDebug->transform.SetPosition(center);
        colliderDebug->transform.SetScale(scale * (radius * 2));

        Debug::Instance().AddDebug(colliderDebug);
    }

private:
    float radius;
};
```

Bileşenler - SimpleGeo

```cpp
void SimpleGeo::Draw(Transform transform)
{
    glPushMatrix();
    glMultMatrixf(transform.ToGLMatrix());
    float halfSize = size / 2.0f;
    glColor3f(color.r, color.g, color.b);
    switch (geoType)
    {
    case TRIANGLE:
        glBegin(GL_TRIANGLES);
        glVertex3f(0.0f, 0.0f, halfSize);
        glVertex3f(-halfSize, 0.0f, -halfSize);
        glVertex3f(halfSize, 0.0f, -halfSize);
        glEnd();
        break;
    case CIRCLE:
    {
        glBegin(GL_TRIANGLE_FAN);
        glVertex3f(0, 0, 0);

        const int stepCount = 64;

        const float twoPI = 2 * 3.14159f;
        for (size_t i = 0; i <= stepCount; i++)
        {
            float t = static_cast<float>(i) / stepCount;
            float angle = t * twoPI;

            float x = cos(angle) * halfSize;
            float z = sin(angle) * halfSize;

            glVertex3f(x, 0, z);
        }
        glEnd();
        break;
    }
```

```cpp
    case CIRCLE_BOUNDS:
    {
        glLineWidth(2.0f);
        glBegin(GL_LINE_LOOP);

        const int stepCount = 64;
        const float twoPI = 2 * 3.14159f;

        for (int i = 0; i < stepCount; i++)
        {
            float t = static_cast<float>(i) / stepCount;
            float angle = t * twoPI;

            float x = cos(angle) * halfSize;
            float z = sin(angle) * halfSize;

            glVertex3f(x, 0, z);
        }

        glEnd();
        break;
    }
    case SQUARE:
        glBegin(GL_QUADS);
        glVertex3f(halfSize, 0.0f, halfSize);
        glVertex3f(halfSize, 0.0f, -halfSize);
        glVertex3f(-halfSize, 0.0f, -halfSize);
        glVertex3f(-halfSize, 0.0f, halfSize);
        glEnd();
        break;
    default:
        break;
    }
```