ПРИЛОЖЕНИЕ Б

Реализации алгоритмов

В листингах 1 – 4 представлены реализации алгоритмов.

Листинг 1 – Алгоритм отрисовки полигона с помощью Z-буффера

```
private void DrawTriangle(Vector3 v1, Vector3 v2, Vector3 v3, float i1,
   float i2, float i3, Color objectColor)
{
    if (v1.Y > v2.Y)
    {
        Swap(ref v1, ref v2);
        Swap(ref i1, ref i2);
    }
    if (v2.Y > v3.Y)
    {
        Swap(ref v2, ref v3);
        Swap(ref i2, ref i3);
    }
    if (v1.Y > v2.Y)
    {
        Swap(ref v1, ref v2);
        Swap(ref i1, ref i2);
    }

    int yStart = (int)Math.Max(0, Math.Ceiling(v1.Y));
    int yEnd = (int)Math.Min(bitmap.Height - 1, Math.Floor(v3.Y));

    for (int y = yStart; y <= yEnd; y++)
    {
        bool secondHalf = y > v2.Y || v2.Y == v1.Y;
        float segmentHeight = secondHalf ? v3.Y - v2.Y : v2.Y - v1.Y;
        if (segmentHeight == 0) segmentHeight = 1;
        float alpha = (y - v1.Y) / (v3.Y - v1.Y);
        float beta = (y - (secondHalf ? v2.Y : v1.Y)) / segmentHeight;

        Vector3 A = v1 + (v3 - v1) * alpha;
        Vector3 B = secondHalf ? v2 + (v3 - v2) * beta : v1 + (v2 - v1)
           * beta;

        float iA = i1 + (i3 - i1) * alpha;
        float iB = secondHalf ? i2 + (i3 - i2) * beta : i1 + (i2 - i1)
           * beta;

        if (A.X > B.X)
        {
            Swap(ref A, ref B);
            Swap(ref iA, ref iB);
        }

        int xStart = (int)Math.Max(0, Math.Ceiling(A.X));
        int xEnd = (int)Math.Min(bitmap.Width - 1, Math.Floor(B.X));

        for (int x = xStart; x <= xEnd; x++)
        {
            float phi = (B.X == A.X) ? 1.0f : (x - A.X) / (B.X - A.X);
            Vector3 P = A + (B - A) * phi;
            float iP = iA + (iB - iA) * phi;

            int zIndex = x;
            int yIndex = y;
            if (zIndex < 0 || zIndex >= bitmap.Width || yIndex < 0 ||
               yIndex >= bitmap.Height)
                continue;

            if (P.Z < zBuffer[zIndex, yIndex])
            {
                zBuffer[zIndex, yIndex] = P.Z;

                int r = (int)(objectColor.R * Clamp(iP, 0, 1));
                int g = (int)(objectColor.G * Clamp(iP, 0, 1));
                int b = (int)(objectColor.B * Clamp(iP, 0, 1));

                bitmap.SetPixel(zIndex, yIndex, Color.FromArgb(r, g,
                   b));
            }
        }
    }
}
```

Листинг 2 – Алгоритм обновления нормалей объекта

```
public void ComputeNormals()
{
    foreach (var vertex in Vertices)
        vertex.Normal = Vector3.Zero;

    foreach (var face in Faces)
    {
        Vector3 v0 = Vertices[face.A].Position;
        Vector3 v1 = Vertices[face.B].Position;
        Vector3 v2 = Vertices[face.C].Position;
        Vector3 edge1 = v1 - v0;
        Vector3 edge2 = v2 - v0;

        Vector3 faceNormal = Vector3.Cross(edge1, edge2);
        if (faceNormal.LengthSquared() > 0)
        {
            faceNormal = Vector3.Normalize(faceNormal);
            Vertices[face.A].Normal += faceNormal;
            Vertices[face.B].Normal += faceNormal;
            Vertices[face.C].Normal += faceNormal;
        }
    }

    foreach (var vertex in Vertices)
    {
        if (vertex.Normal.LengthSquared() > 0)
            vertex.Normal = Vector3.Normalize(vertex.Normal);
        else
            vertex.Normal = Vector3.UnitY; // Default normal
    }
}
```

Листинг 3 – Реализация модели освещения Ламберта

```
private float ComputeLighting(Vector3 position, Vector3 normal)
{
    float intensity = 0;
    float ambient = 0.2f * light.Intensity;
    intensity += ambient;
    Vector3 lightDir = Vector3.Normalize(light.Position - position);
    float nDotL = Vector3.Dot(normal, lightDir);

    if (nDotL > 0)
    {
        if (IsInShadow(position, light.Position))
            intensity = ambient;
        else
            intensity += nDotL * light.Intensity;
    }

    return Clamp(intensity, 0, 1);
}
```

Листинг 4 – Реализация алгоритма проверки затенения точки объектом

```
private bool IsInShadow(Vector3 point, Vector3 lightPos)
{
    Vector3 dir = Vector3.Normalize(lightPos - point);
    float distanceToLight = Vector3.Distance(lightPos, point);
    if (distanceToLight < 0.001f)
        return false;
    float bias = 0.001f;
    Vector3 shadowOrigin = point + dir * bias;

    foreach (var mesh in scene.Meshes)
    {
        Matrix4x4 worldMatrix =
           Matrix4x4.CreateFromQuaternion(mesh.Rotation) *
           Matrix4x4.CreateTranslation(mesh.Position);

        foreach (var face in mesh.Faces)
        {
            Vertex v1 = mesh.Vertices[face.A];
            Vertex v2 = mesh.Vertices[face.B];
            Vertex v3 = mesh.Vertices[face.C];
            Vector3 worldV1 = Vector3.Transform(v1.Position,
               worldMatrix);
            Vector3 worldV2 = Vector3.Transform(v2.Position,
               worldMatrix);
            Vector3 worldV3 = Vector3.Transform(v3.Position,
               worldMatrix);
            if (IntersectTriangle(shadowOrigin, dir, worldV1, worldV2,
               worldV3, out float t))
                if (t > 0 && t < distanceToLight)
                    return true;
        }
    }
    return false;
}

private bool IntersectTriangle(Vector3 orig, Vector3 dir, Vector3 v0,
   Vector3 v1, Vector3 v2, out float t)
{
    t = 0;
    const float EPSILON = 0.0000001f;
    Vector3 edge1 = v1 - v0;
    Vector3 edge2 = v2 - v0;
    Vector3 h = Vector3.Cross(dir, edge2);
    float a = Vector3.Dot(edge1, h);
    if (a > -EPSILON && a < EPSILON)
        return false;
    float f = 1.0f / a;
    Vector3 s = orig - v0;
    float u = f * Vector3.Dot(s, h);
    if (u < 0.0 || u > 1.0)
        return false;
    Vector3 q = Vector3.Cross(s, edge1);
    float v = f * Vector3.Dot(dir, q);
    if (v < 0.0 || u + v > 1.0)
        return false;
    t = f * Vector3.Dot(edge2, q);
    if (t > EPSILON)
        return true;
    else
        return false;
}
```