Компьютерная графика. Часть 2.

Лектор – доц. Беляев Сергей Юрьевич.

Оглавление

1. Организация рендера

1.1. Определение видимых объектов

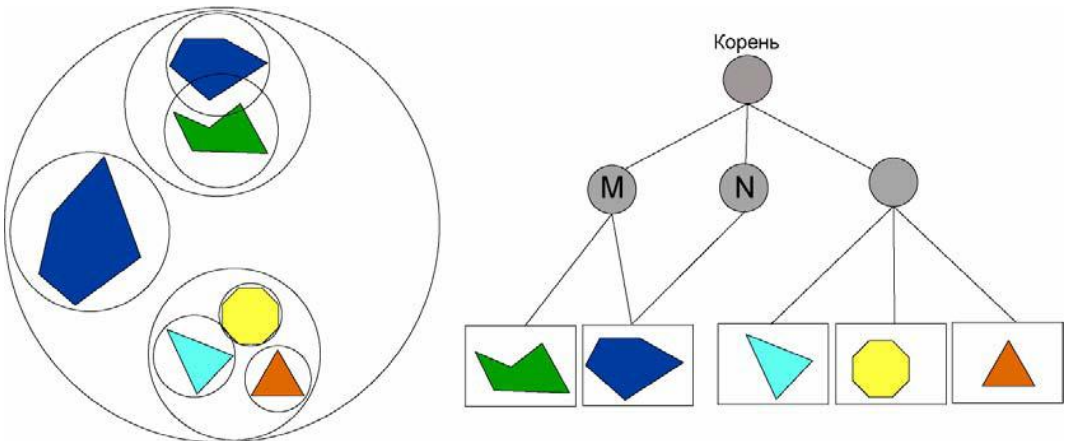


Рис. 1.1 Иерархия ограничивающих объемов и граф сцены

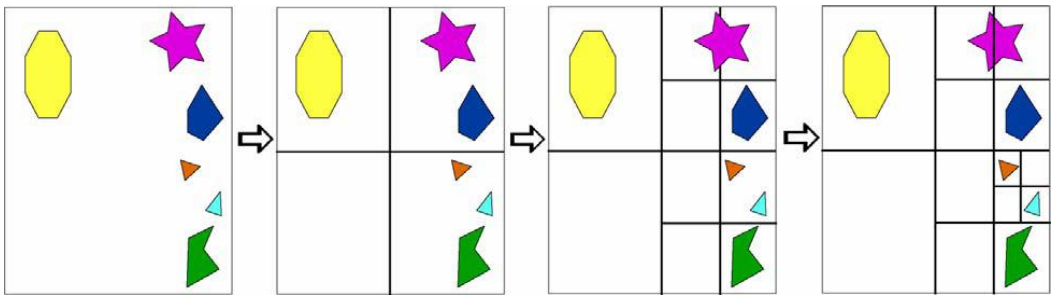


Рис. 1.2 Октарное дерево (octree)

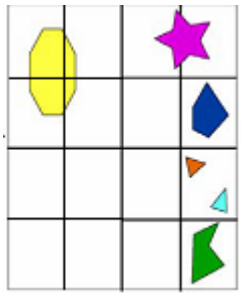


Рис. 1.3 Воксельная сетка (Voxel)

1.2. Отбрасывание объектов закрытых другими объектами (Occlusion culling)

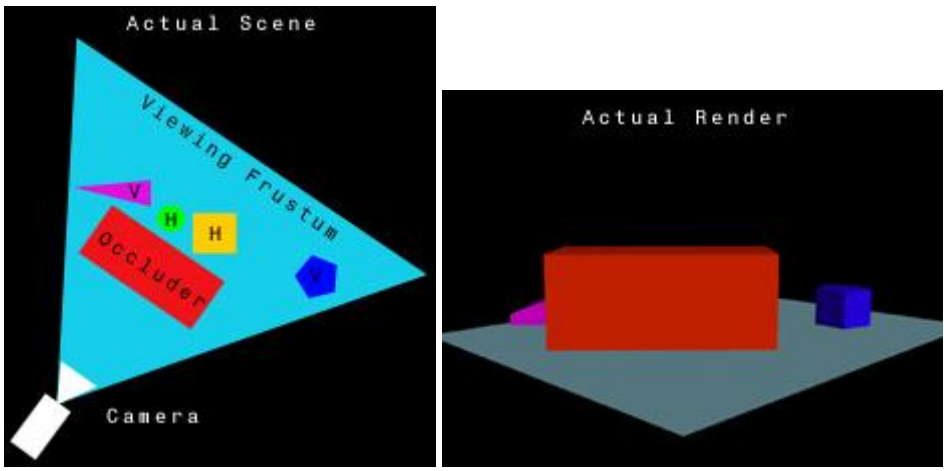


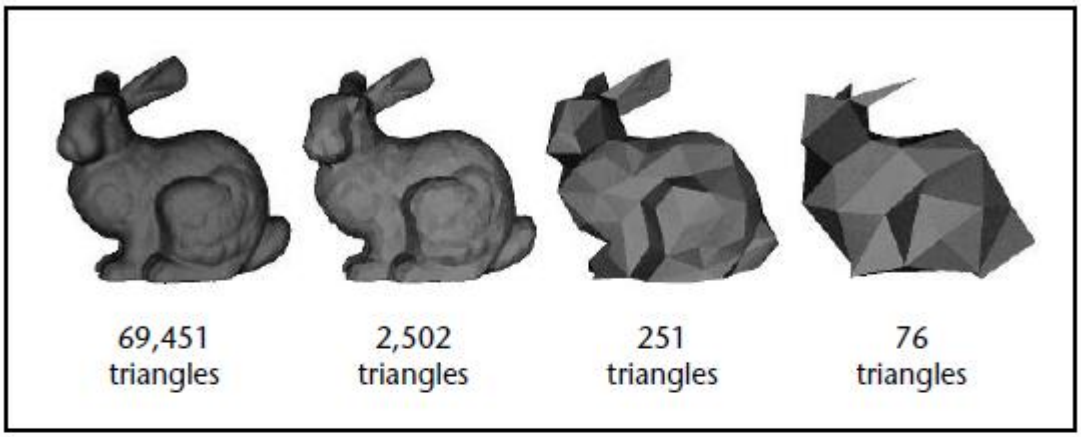
Рис. 1.4

Occlusion query

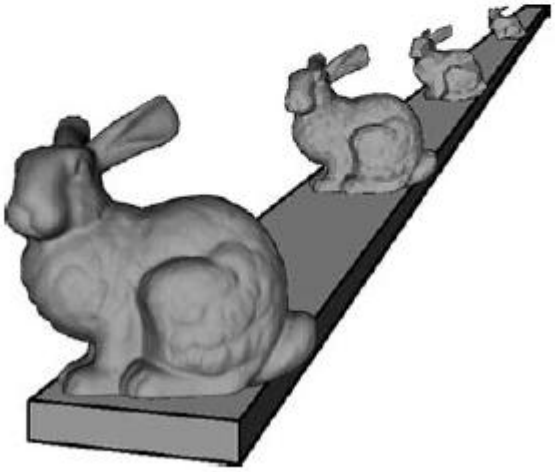
1. Упрощенный render сцены => Z-буфер
2. For every object:
    a. Begin query
    b. render the bounding mesh
    c. End query
    d. Retrieve occlusion query data. If the pixels visible are greater than zero, the object should be rendered. Otherwise, the object should be occluded from rendering.

1.3. Детализация объектов сцены

Уровни детализации (LOD)



(a)



(b)

Рис. 1.5

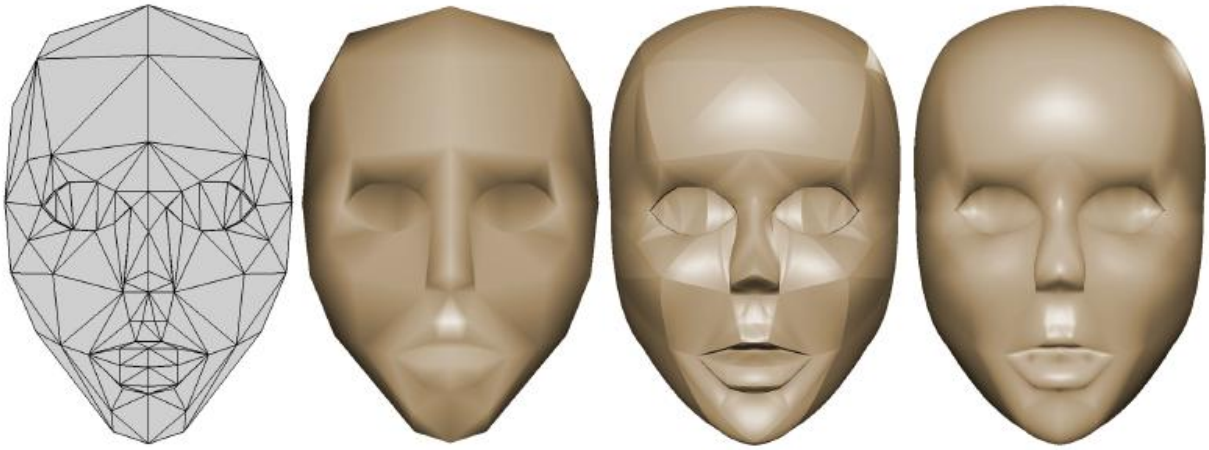


Рис. 1.6 Tessellation

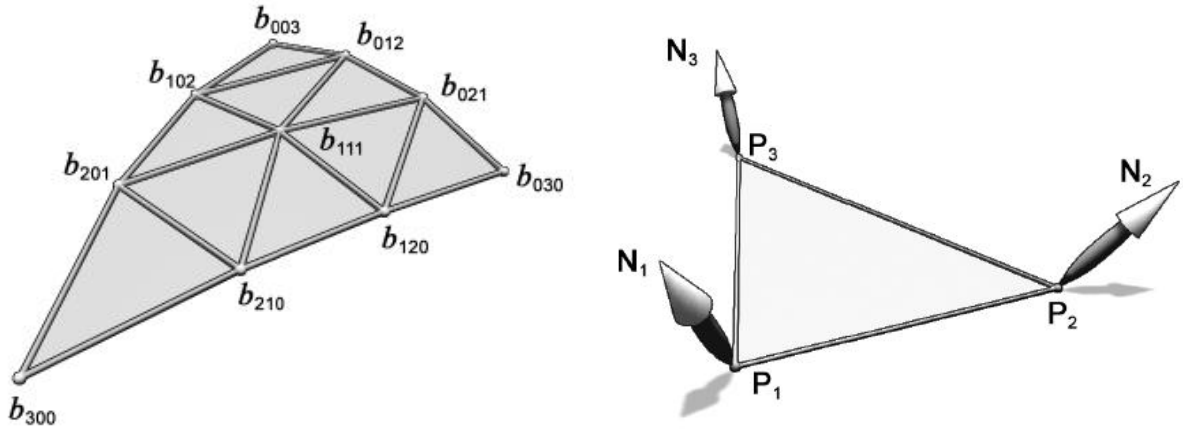


Рис. 1.7 PN Triangles

$$
\begin{aligned}
w_{ij} &= (P_j - P_i) \cdot N_i \in \mathbf{R} \\
b_{210} &= (2P_1 + P_2 - w_{12}N_1)/3, \\
b_{120} &= (2P_2 + P_1 - w_{21}N_2)/3, \\
b_{021} &= (2P_2 + P_3 - w_{23}N_2)/3, \\
b_{012} &= (2P_3 + P_2 - w_{32}N_3)/3, \\
b_{102} &= (2P_3 + P_1 - w_{31}N_3)/3, \\
b_{201} &= (2P_1 + P_3 - w_{13}N_1)/3, \\
E &= (b_{210} + b_{120} + b_{021} + b_{012} + b_{102} + b_{201})/6 \\
V &= (P_1 + P_2 + P_3)/3, \\
b_{111} &= E + (E - V)/2.
\end{aligned}
$$

$P = uP_1 + vP_2 + wP_3$, $w=1-u-v$, $u+v+w=1$

$P = P_3 + u(P_1 - P_3) + v(P_2 - P_3)$

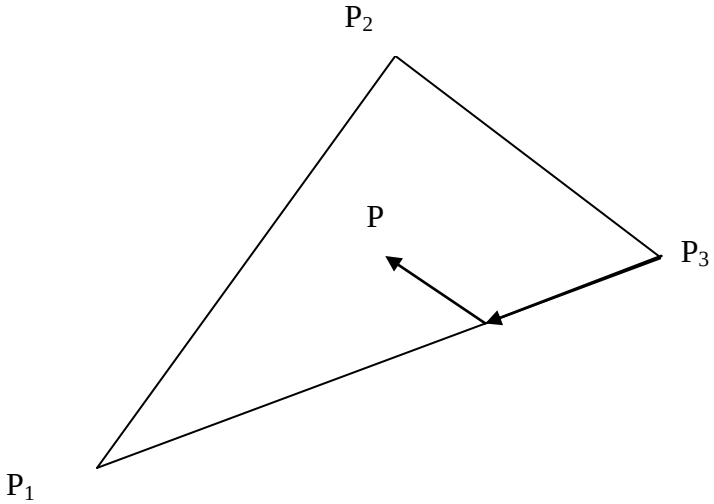


Рис. 1.8

$$
\begin{aligned}
b(u,v) &= \sum_{i+j+k=3} b_{ijk} \frac{3!}{i!j!k!} u^i v^j w^k, \\
&= b_{300} w^3 + b_{030} u^3 + b_{003} v^3 \\
&\quad + b_{210} 3w^2 u + b_{120} 3wu^2 + b_{201} 3w^2 v \\
&\quad + b_{021} 3u^2 v + b_{102} 3wv^2 + b_{012} 3uv^2 \\
&\quad + b_{111} 6wuv.
\end{aligned}
$$

2. Shaders

2.1 Direct3D Pipeline

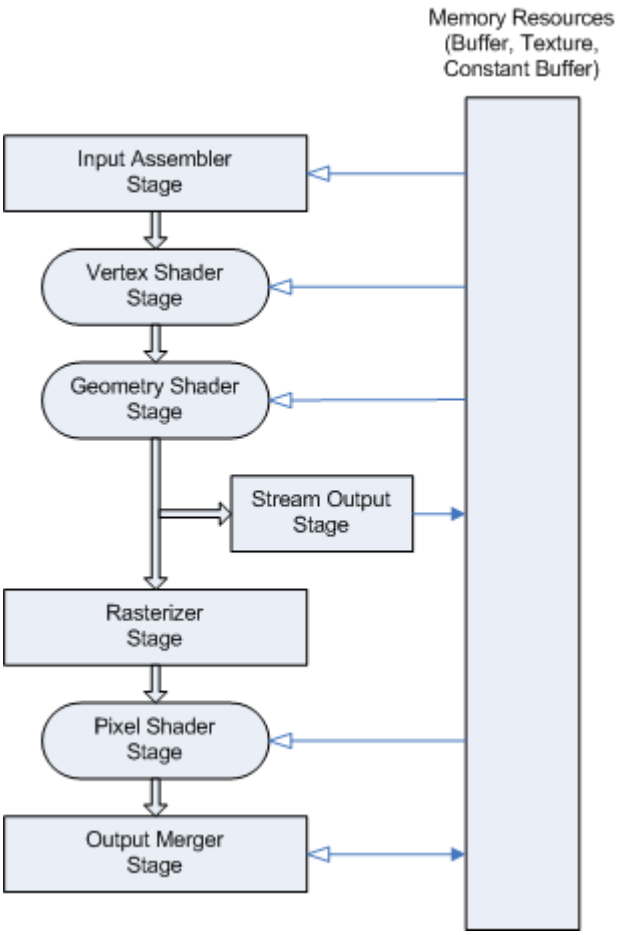


Рис. 2.1

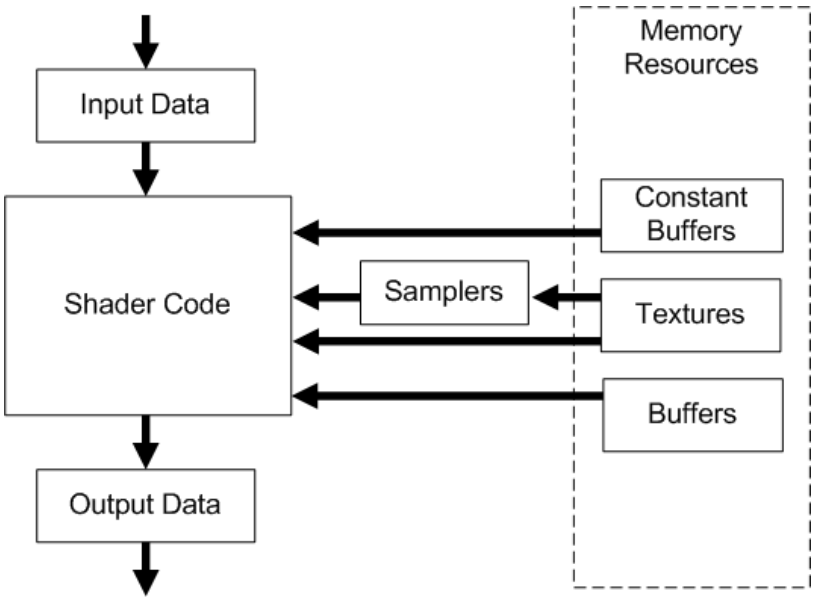


Рис. 2.2

2.2 Vertex shader

Расчет освещения

Vertex input (например):

- Position
- Normal
- Texture coordinate (n)

Constant Buffer

W, WV, Light direction, Ambient, Deffuse, Specular

Vertex Output

- Position
- Color (diffuse)
- Color (specular)
- Texture coordinate (n)

Скелетная анимация

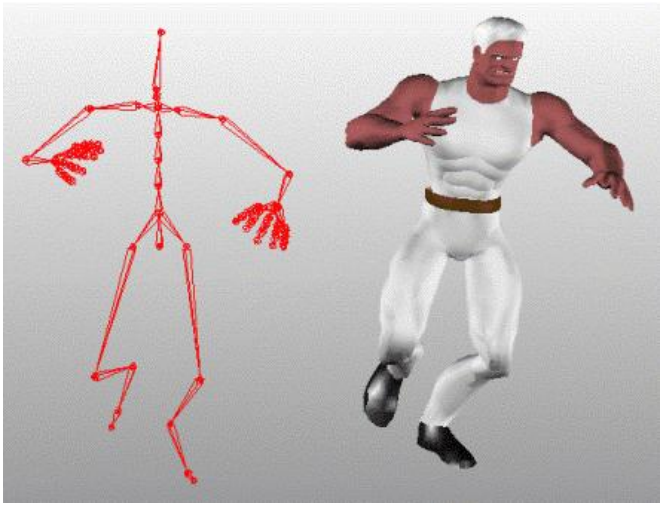
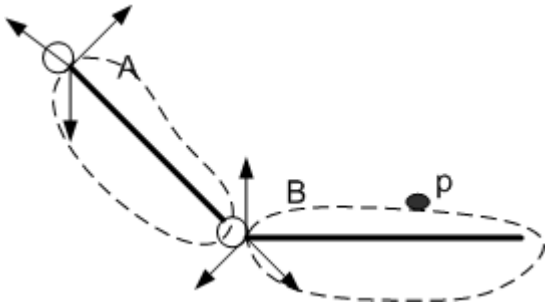


Рис. 2.2

Joint - точка, вокруг которой происходит трансформация любых привязанных к ней объектов.
Скелет - набор joint'ов, связанных между собой по принципу "родитель"-"потомок", причем преобразования каждого joint'a заданы в системе координат "родительского" joint'a.



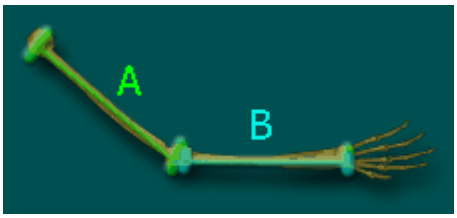


Рис. 2.3

$p_A = p*B$
$p_w = p*L_n*L_{n-1}*\ldots*L_1$
$W_{n-1} = L_{n-1}*\ldots*L_1$
$p_w = p*L_n*W_{n-1}$

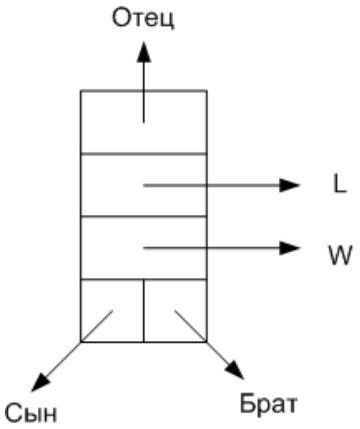


Рис. 2.4

Animation sequence
L: $L^1, L^2\ldots, L^m$

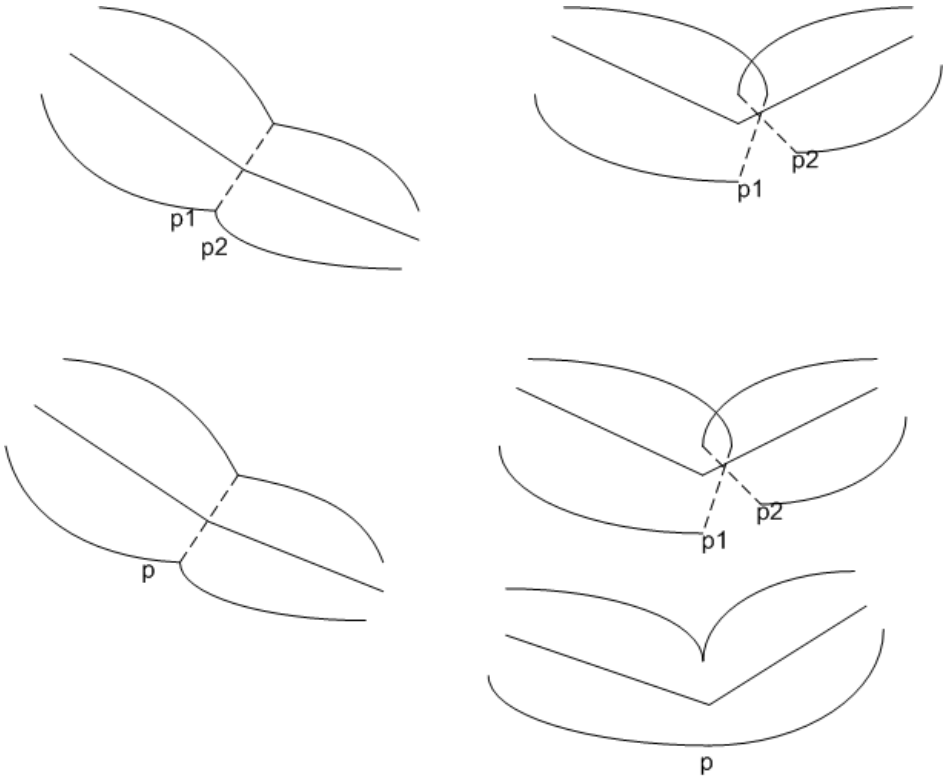


Рис. 2.4 Skin

p1 = p*W1
p2 = p*W2
p = b*p1 + (1-b)*p2

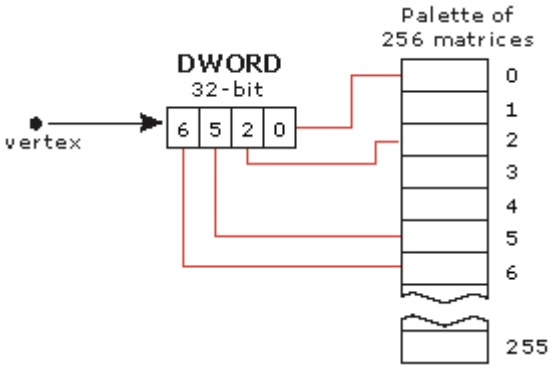


Рис. 2.5

$$V_{world} = V_{model} * M[index0] * b_0 + V_{model} * M[index1] * b_1 + V_{model} * M[index2] * b_2 + V_{model} * M[index3] * (1 - b_0 - b_1 - b_2)$$

```
struct BLENDVERTEX
{
    D3DXVECTOR3 v;        // Referenced as v0 in the vertex shader
    FLOAT       blend1;   // Referenced as v1.x in the vertex shader
    FLOAT       blend2;   // Referenced as v1.y in the vertex shader
    FLOAT       blend3;   // Referenced as v1.z in the vertex shader
                          // v1.w = 1.0 - (v1.x + v1.y + v1.z)
    DWORD       indices;  // Referenced as v2.xyzw in the vertex shader
    D3DXVECTOR3 n;        // Referenced as v3 in the vertex shader
    FLOAT       tu, tv;   // Referenced as v7 in the vertex shader
};
```

2.3 Input assembler stage

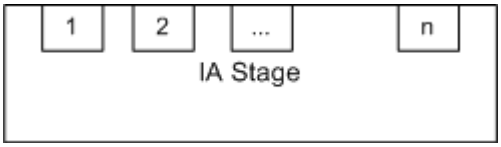


Рис. 2.6 Input Slots (stream)

Position копии (для анимации)

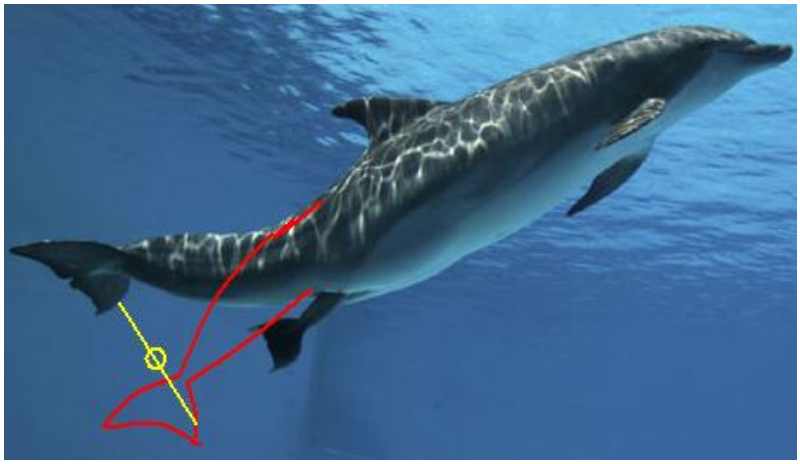


Рис. 2.7 $\mathbf{p}=t*\mathbf{p}_1+(1-t)\mathbf{p}_2$

Рис. 2.8 Instancing

Instancing the Tree
Slot1 – TreeVertexBuff
Slot2 – TreeMatrixBuff

Instancing the Leaves

Slot1 – LeaveVertexBuff
Slot2 – LeavesMatrixBuff
Constant buffer - TreeMatrixBuff

```
Tree1, leaf1; Tree2, leaf1; Tree3, leaf1; Tree1, leaf2; Tree2, leaf2; etc...

int iTree = input.InstanceId%g_iNumTrees;
    float4 vInstancePos = mul( float4(input.pos, 1), input.mTransform  );
    float4 InstancePosition = mul(vInstancePos, g_mTreeMatrices[iTree] );
```

2.4 Geometry shader

Вход – треугольник. Выход – множество треугольников.
Трава

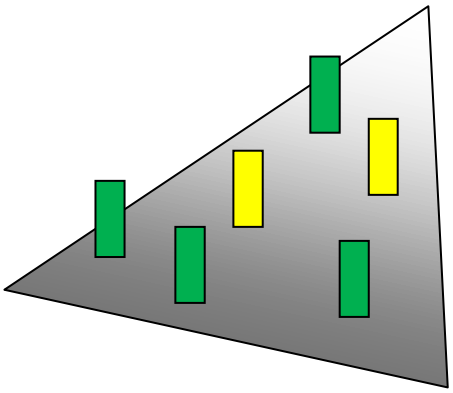


Рис. 2.9

PrimitiveID -> 1D текстура -> Псевдо случайные числа->позиции
Текстурные координаты
Массив текстур
Частицы

Рис. 2.10

p - позиция
v - скорость
t - время видимости
c – счетчик
Buff1

p, v, t						
0	-1	-2	-3	-4	……	-n

Buff2

p, v, t						
					……	

Buff[PrimetiveID].**p**
If (c==0) случайные **p, v**, t<n
Else
{
v+=**F***dt;
p+=**v***dt;
}
c++;
c%=n;
Первый рендер:
GS OutStream: Buff1

p0, v0, t						
1	0	-1	-2	-3	……	-n+1

If (Buff[PrimetiveID].c>0&& Buff[PrimetiveID].t<n) GS OutPut: 
Buff2

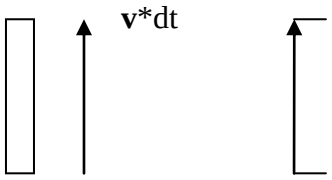
					……	

Второй рендер:
Buff1

p, v, t						
1	0	-1	-2	-3	……	-n+1

Buff2

p1, v1, t	**p0, v0**, t					
2	1	0	-1	-2	……	-n+2



2.5 Hull/Domain shaders

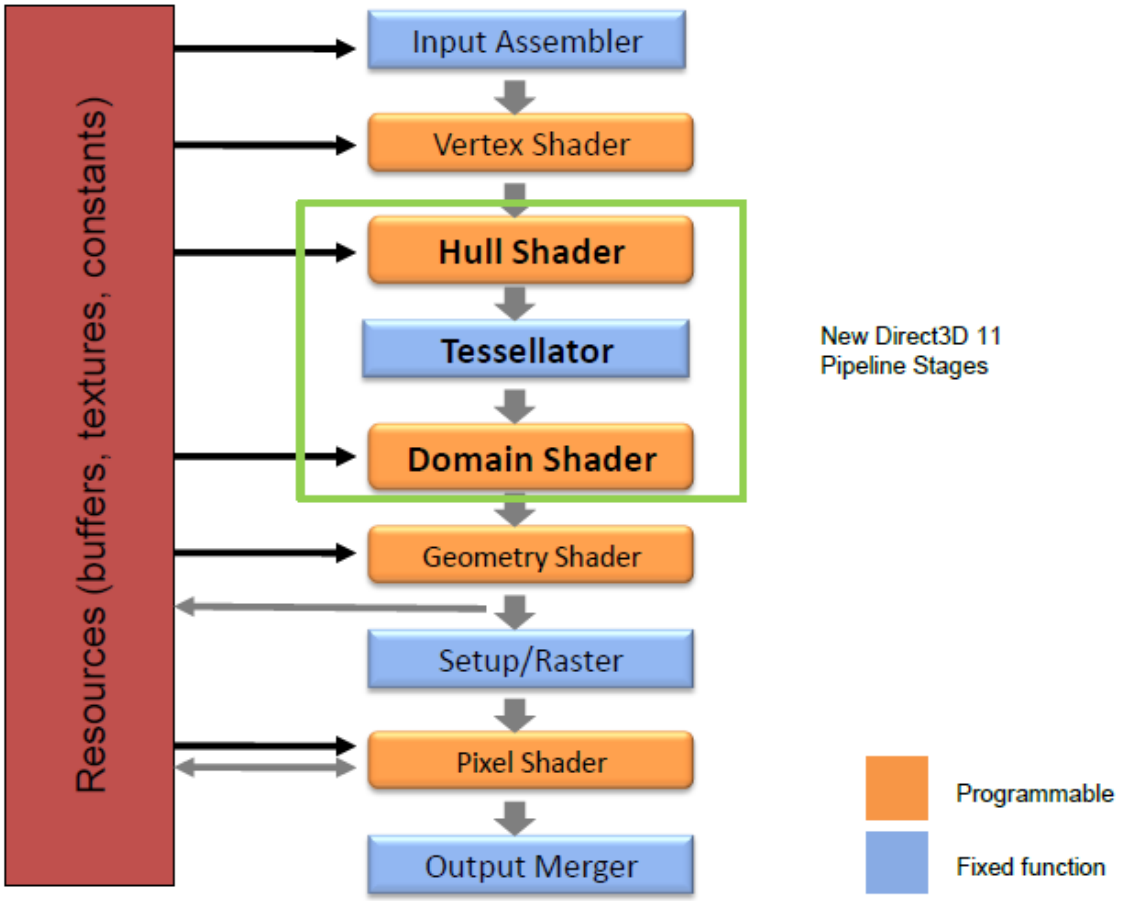

Resources (buffers, textures, constants)
Input Assembler
Vertex Shader
Hull Shader
Tessellator
Domain Shader
Geometry Shader
Setup/Raster
Pixel Shader
Output Merger
New Direct3D 11
Pipeline Stages
Programmable
Fixed function

Рис. 2.11

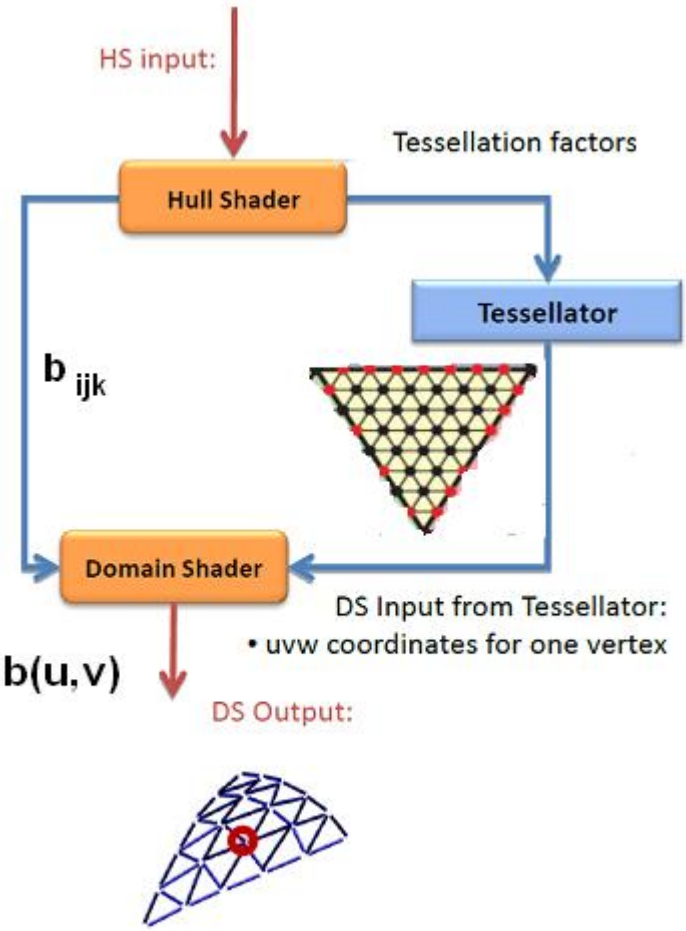


Рис. 2.12

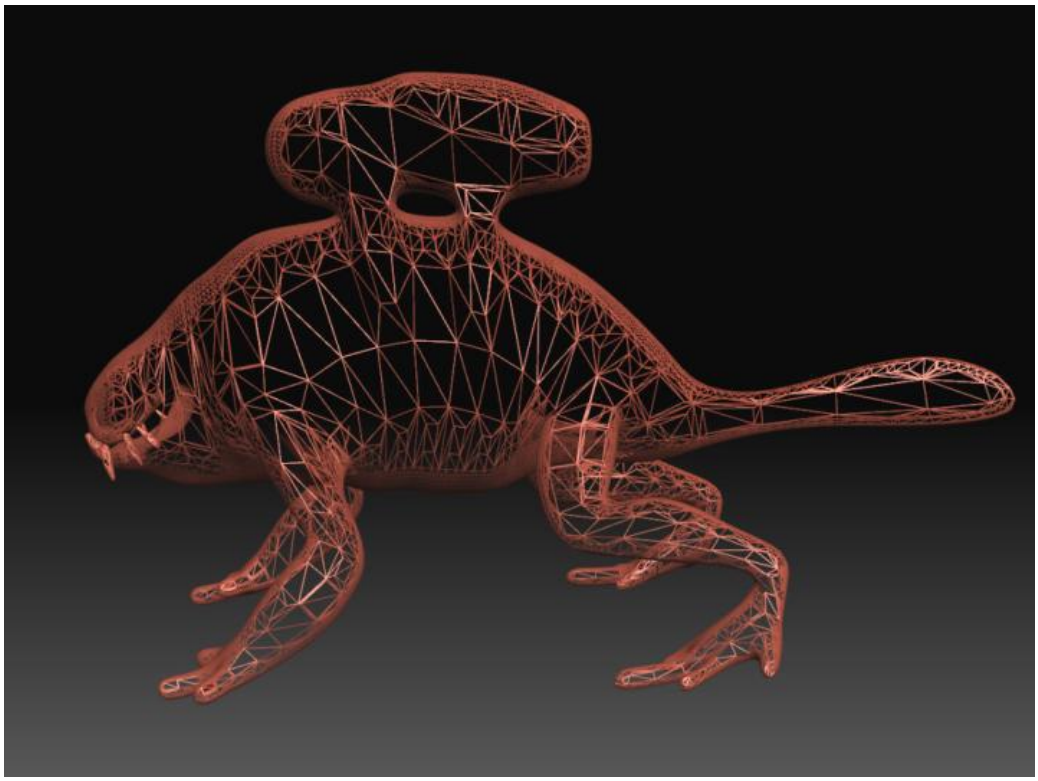


Рис. 2.13 Адаптивная тесселяция

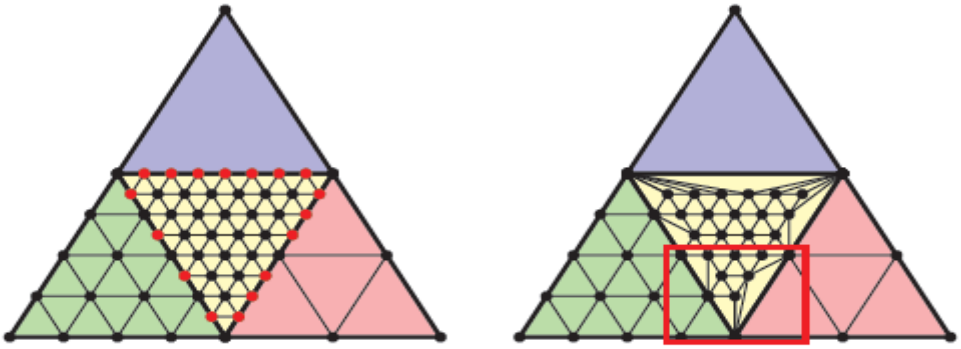


Рис. 2.14

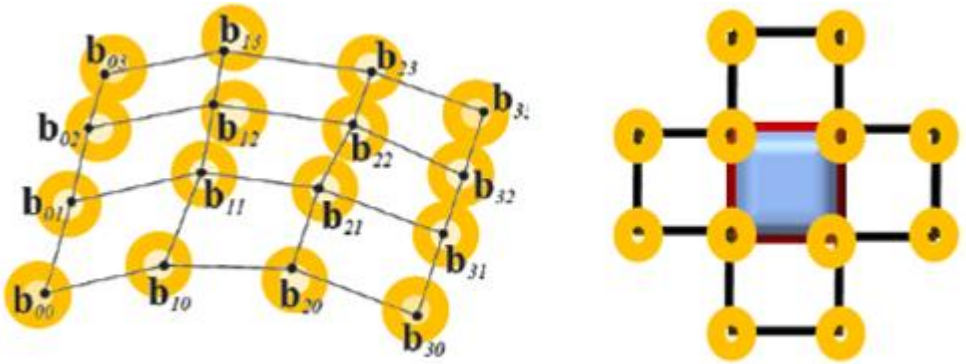


Рис. 2.15

2.6 Pixel shader

Input:

- Color (diffuse)
- Color (specular)
- Texture coordinates (n)

Output:

- Color (m)
- Depth (m)

Пример. Закраска по Фонгу.
Vertex shader output:

- VertexPos (Камерная система координат)
- Normal

Pixel shader input: то же в пикселе
Pixel shader constats:

- Позиция и другие параметры источника
- Параметры материала

L= Позиция источника - VertexPos
(N,L)
D=|L|

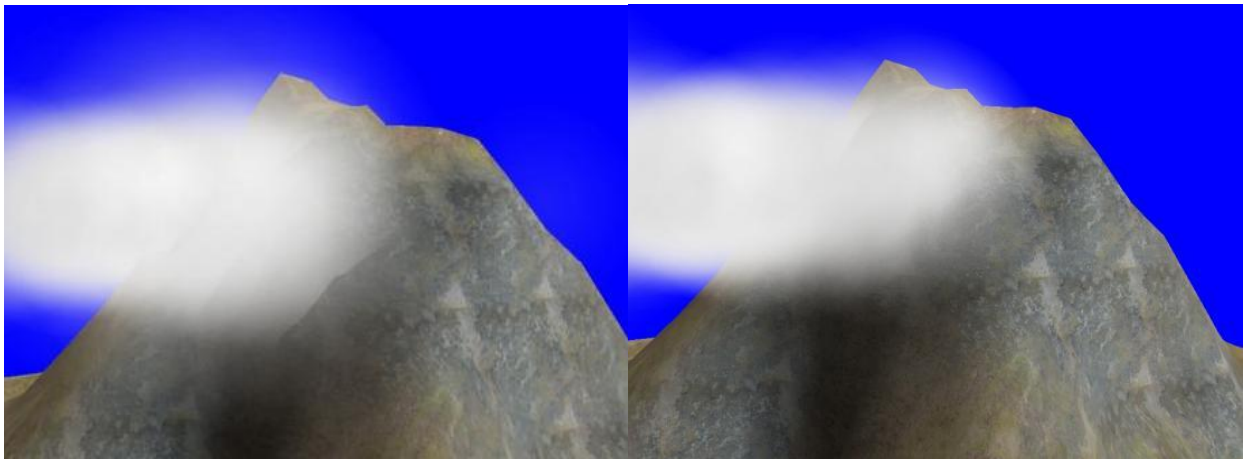


Рис. 2.16 Soft particles

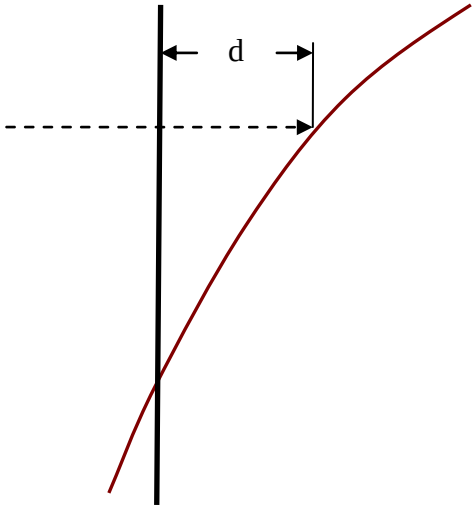


Рис. 2.17

Прозрачность = F(d); F(d)=0 при d<=0