1 Introduction

If we render textures using bi-linear interpolation we will observe aliasing artifacts in ray-traced images. Intuitively, we can avoid such problems by averaging the texture over the area of a pixel instead of evaluating it at a single point. To implement such a procedure we have to solve two problems: First, we need to compute the region in the texture that corresponds to the pixel area. We call this region the pixel footprint. We describe pixel footprint computation in Section 2. Second, we need a fast procedure to obtain the average of the texture over this area. This can be achieved using mip-mapping and trilinear interpolation, which is described in Section 3.

2 Calculating the Pixel Footprint

2.1 Overview

We denote pixel coordinates by $x$ and $y$. We represent the area of a rectangular pixel with side lengths $\Delta x$ and $\Delta y$ by two perpendicular vectors

$$
\begin{bmatrix}
\Delta x \\
0
\end{bmatrix}
\quad \text{and} \quad
\begin{bmatrix}
0 \\
\Delta y
\end{bmatrix}
$$

(1)

Our goal is to find the two corresponding vectors in texture space that span the area corresponding to the pixel. Let us assume we know the functions $u(x, y)$ and $v(x, y)$ that map pixel coordinates to texture coordinates. In other words, given a ray through pixel $x, y$, these functions return the texture coordinates $u, v$ at the ray-surface intersection point. The Jacobi matrix of these functions is defined as the $2 \times 2$ matrix consisting of their partial derivatives. Intuitively, the Jacobi matrix relates small changes in pixel coordinates to small changes in texture coordinates:

$$
\begin{bmatrix}
\Delta u \\
\Delta v
\end{bmatrix} = \begin{bmatrix}
\frac{\partial u(x, y)}{\partial x} & \frac{\partial u(x, y)}{\partial y} \\
\frac{\partial v(x, y)}{\partial x} & \frac{\partial v(x, y)}{\partial y}
\end{bmatrix} \begin{bmatrix}
\Delta x \\
\Delta y
\end{bmatrix}.
$$

We multiply the Jacobi matrix with the vectors spanning a rectangular pixel from Equation 1. This yields the vectors in texture coordinates that span the
area corresponding to the pixel:
\[
\begin{bmatrix}
\frac{\partial u(x, y)}{\partial x} & \frac{\partial u(x, y)}{\partial y} \\
\frac{\partial v(x, y)}{\partial x} & \frac{\partial v(x, y)}{\partial y}
\end{bmatrix} \begin{bmatrix} \Delta x \\ 0 \end{bmatrix} = \begin{bmatrix} \frac{\partial u(x, y)}{\partial x} \Delta x \\ \frac{\partial v(x, y)}{\partial y} \Delta y \end{bmatrix}.
\]

The pixel footprint in texture coordinates is given by the parallelogram spanned by these two vectors.

### 2.2 Detailed Derivation

We now describe in more detail how to compute the partial derivatives $\frac{\partial u}{\partial x}$, $\frac{\partial u}{\partial y}$, $\frac{\partial v}{\partial x}$, and $\frac{\partial v}{\partial y}$. Instead of deriving the functions $u(x, y)$ and $v(x, y)$, we will first compute the inverse functions $x(u, v)$ and $y(u, v)$ and their Jacobi matrix. The desired partial derivatives are then found by inverting this Jacobi matrix. Our discussion here is restricted to the case of triangles with texture coordinates.

The functions $x(u, v)$ and $y(u, v)$ are a concatenation of several steps, which we will consider separately. First, we will derive a function $p(u, v)$ that computes a 3D point on a triangle given a pair of texture coordinates. Then we will introduce functions $x'(p)$ and $y'(p)$ that project 3D points to pixel coordinates. The concatenation of these two steps yields the desired functions $x(u, v) = x'(p(u, v))$ and $y(u, v) = y'(p(u, v))$.

Let us denote the three coordinates of a point $p$ in world space by $p_0, p_1, p_2$. For each of the coordinates we will determine a linear mapping $p_0, p_1, p_2$ given by three coefficients. We write the linear mappings with unknown coefficients $a, b, c \ldots i$

\[
p_0(u, v) = \begin{bmatrix} u \\ v \\ 1 \end{bmatrix} \begin{bmatrix} a \\ b \\ c \end{bmatrix},
\]

\[
p_1(u, v) = \begin{bmatrix} u \\ v \\ 1 \end{bmatrix} \begin{bmatrix} d \\ e \\ f \end{bmatrix},
\]

\[
p_2(u, v) = \begin{bmatrix} u \\ v \\ 1 \end{bmatrix} \begin{bmatrix} g \\ h \\ i \end{bmatrix}.
\]

Given that each triangle vertex stores 3D world coordinates $p_{0,i}, p_{1,i}, p_{2,i}$ and texture coordinates $u_i, v_i$ for $i \in \{0, 1, 2\}$, we can formulate three equations with three unknowns for each of the coordinates:

\[
\begin{bmatrix} p_{0,0} \\ p_{0,1} \\ p_{0,2} \end{bmatrix} = \begin{bmatrix} u_0 \\ u_1 \\ u_2 \end{bmatrix} \begin{bmatrix} 0 \\ 1 \\ 1 \end{bmatrix} \begin{bmatrix} a \\ b \\ c \end{bmatrix}.
\]
\[
\begin{bmatrix}
  p_{1,0} \\
p_{1,1} \\
p_{1,2}
\end{bmatrix} = \begin{bmatrix}
  u_0 & v_0 & 1 \\
u_1 & v_1 & 1 \\
u_2 & v_2 & 1
\end{bmatrix} \begin{bmatrix}
d \\
e \\
f
\end{bmatrix},
\]
\[
\begin{bmatrix}
p_{2,0} \\
p_{2,1} \\
p_{2,2}
\end{bmatrix} = \begin{bmatrix}
  u_0 & v_0 & 1 \\
u_1 & v_1 & 1 \\
u_2 & v_2 & 1
\end{bmatrix} \begin{bmatrix}
g \\
h \\
i
\end{bmatrix}.
\]

Therefore, the unknown coefficients are given by
\[
\begin{bmatrix}
a \\
b \\
c
\end{bmatrix} = \begin{bmatrix}
u_0 & v_0 & 1 \\
u_1 & v_1 & 1 \\
u_2 & v_2 & 1
\end{bmatrix}^{-1} \begin{bmatrix}
p_{0,0} \\
p_{0,1} \\
p_{0,2}
\end{bmatrix},
\]
\[
\begin{bmatrix}
d \\
e \\
f
\end{bmatrix} = \begin{bmatrix}
u_0 & v_0 & 1 \\
u_1 & v_1 & 1 \\
u_2 & v_2 & 1
\end{bmatrix}^{-1} \begin{bmatrix}
p_{1,0} \\
p_{1,1} \\
p_{1,2}
\end{bmatrix},
\]
\[
\begin{bmatrix}
g \\
h \\
i
\end{bmatrix} = \begin{bmatrix}
u_0 & v_0 & 1 \\
u_1 & v_1 & 1 \\
u_2 & v_2 & 1
\end{bmatrix}^{-1} \begin{bmatrix}
p_{2,0} \\
p_{2,1} \\
p_{2,2}
\end{bmatrix}.
\]

This means that we need to perform one inversion of a $3 \times 3$ matrix and three matrix-vector multiplications to find the unknown coefficients.

We now have the mapping from texture coordinates to a point in 3D world coordinates
\[
p(u, v) = \begin{bmatrix}
p_0(u, v) \\
p_1(u, v) \\
p_2(u, v)
\end{bmatrix} = \begin{bmatrix}
a & b & c \\
d & e & f \\
g & h & i
\end{bmatrix} \begin{bmatrix}
u \\
v \\
1
\end{bmatrix} = \mathbf{M}_{uv \rightarrow w} \begin{bmatrix}
u \\
v \\
1
\end{bmatrix},
\]
where we expressed $p$ using homogeneous coordinates and introduced the matrix $\mathbf{M}_{uv \rightarrow w}$ containing the coefficients $a, b, c, \ldots i$.

Next, we look at the projection of the homogeneous point $p$ from world coordinates to $x, y$ image coordinates, which we represent by the functions $x'(p)$ and $y'(p)$. This involves the transformation of $p$ to camera coordinates, the perspective projection, and 2D scaling and translation to image coordinates. Let us denote the $4 \times 4$ transformation from world to camera coordinates by $\mathbf{M}_{w \rightarrow c}$. The projection functions $x'(p)$ and $y'(p)$ are given by
\[
x'(p) = \frac{(\mathbf{M}_{w \rightarrow c} p)_0}{(\mathbf{M}_{w \rightarrow c} p)_2} s_x + t_x,
\]
\[
y'(p) = \frac{(\mathbf{M}_{w \rightarrow c} p)_1}{(\mathbf{M}_{w \rightarrow c} p)_2} s_y + t_y,
\]
where $s_x, t_x, s_y, t_y$ are scaling and translation coefficients that are given by the field of view of your camera and the image resolution. Note that the notation $(\cdot)_0, (\cdot)_1, (\cdot)_2$ refers to the elements 0, 1, 2 of a given vector.
We now express \( \mathbf{p} \) using Equation 7 and introduce the \( 4 \times 3 \) matrix \( \mathbf{M} = M_{w \rightarrow c} M_{uv \rightarrow w} \) whose elements we call \( m_{i,j} \). We rewrite Equations 8 and 9 as

\[
x(u, v) = \frac{m_{0,0}u + m_{0,1}v + m_{0,2}}{m_{2,0}u + m_{2,1}v + m_{2,2}} s_x + t_x,
\]

\[
y(u, v) = \frac{m_{1,0}u + m_{1,1}v + m_{1,2}}{m_{2,0}u + m_{2,1}v + m_{2,2}} s_y + t_y.
\]

We use the quotient rule to obtain the partial derivatives of \( x(u, v) \) and \( y(u, v) \):

\[
\frac{\partial x}{\partial u} = \frac{m_{0,0} \Delta x + m_{0,1} \Delta y}{(m_{2,0}u + m_{2,1}v + m_{2,2})^2} - \frac{m_{0,1} \Delta x}{(m_{2,0}u + m_{2,1}v + m_{2,2})^2}, \tag{10}
\]

\[
\frac{\partial x}{\partial v} = \frac{m_{0,1} \Delta x + m_{0,1} \Delta y}{(m_{2,0}u + m_{2,1}v + m_{2,2})^2} - \frac{m_{0,0} \Delta y}{(m_{2,0}u + m_{2,1}v + m_{2,2})^2}, \tag{11}
\]

\[
\frac{\partial y}{\partial u} = \frac{m_{1,0} \Delta x + m_{1,1} \Delta y}{(m_{2,0}u + m_{2,1}v + m_{2,2})^2} - \frac{m_{1,1} \Delta x}{(m_{2,0}u + m_{2,1}v + m_{2,2})^2}, \tag{12}
\]

\[
\frac{\partial y}{\partial v} = \frac{m_{1,1} \Delta x + m_{1,1} \Delta y}{(m_{2,0}u + m_{2,1}v + m_{2,2})^2} - \frac{m_{1,0} \Delta y}{(m_{2,0}u + m_{2,1}v + m_{2,2})^2}. \tag{13}
\]

These partial derivatives are the elements of the Jacobian matrix \( \mathbf{J} \) of \( x(u, v) \) and \( y(u, v) \):

\[
\mathbf{J} = \begin{bmatrix}
\frac{\partial x(u,v)}{\partial u} & \frac{\partial x(u,v)}{\partial v} \\
\frac{\partial y(u,v)}{\partial u} & \frac{\partial y(u,v)}{\partial v}
\end{bmatrix}. \tag{14}
\]

Finally, we compute the inverse of this matrix to obtain the partial derivatives that we were looking for

\[
\mathbf{J}^{-1} = \begin{bmatrix}
\frac{\partial u(x,y)}{\partial x} & \frac{\partial u(x,y)}{\partial y} \\
\frac{\partial v(x,y)}{\partial x} & \frac{\partial v(x,y)}{\partial y}
\end{bmatrix}. \tag{15}
\]

Note that we are usually interested in the footprint of unit square pixels, i.e., \( \Delta x = 1 \) and \( \Delta y = 1 \) in Equations 2 and 3. Therefore, the pixel footprint in texture coordinates is spanned by the vectors

\[
\begin{bmatrix}
\frac{\partial u(x,y)}{\partial x} \\
\frac{\partial v(x,y)}{\partial x}
\end{bmatrix}, \quad \text{and} \quad \begin{bmatrix}
\frac{\partial u(x,y)}{\partial y} \\
\frac{\partial v(x,y)}{\partial y}
\end{bmatrix}. \tag{16}
\]

### 2.3 Implementation

A practical implementation of the procedure described in the last section proceeds as follows:

- In your ray-triangle intersection procedure, compute the matrix \( \mathbf{M}_{uv \leftarrow p} \), which was introduced in Equation 7, by evaluating Equations 4, 5, and 6.
- Next, compute the matrix \( \mathbf{M} = \mathbf{M}_{w \rightarrow c} \mathbf{M}_{uv \rightarrow w} \). The world-to-camera transformation matrix \( \mathbf{M}_{w \rightarrow c} \) should be included in the ray data structure so that it is available in the ray-triangle intersection procedure.
Now evaluate the partial derivatives given in Equations 10, 11, 12, and 13. The scaling coefficients $s_x$ and $s_y$ should also be passed to the ray-triangle intersection procedure through the ray data structure.

Finally, arrange the partial derivatives as the $2 \times 2$ Jacobi matrix $J$ shown in Equation 14. Invert this matrix, which yields the desired partial derivatives as in Equation 15.

3 Mip-mapping

For mip-mapping we simply approximate the pixel footprint by a square. The sidelength of the square is chosen as

$$w = \max \left( \left\| \begin{bmatrix} \frac{\partial u(x,y)}{\partial x} \\ \frac{\partial u(x,y)}{\partial y} \end{bmatrix} \right\|, \left\| \begin{bmatrix} \frac{\partial v(x,y)}{\partial x} \\ \frac{\partial v(x,y)}{\partial y} \end{bmatrix} \right\| \right). \quad (17)$$

Note that the width of the pixel footprint here is expressed in canonic texture coordinates that are in the range $[0, 1] \times [0, 1]$.

Given a square texture with resolution $2^{n-1}$, we construct a mip-map pyramid with $n$ levels where the resolution of level $l$ is $2^{n-1-l}$. For a square footprint of width $w$ the mip-map level $l$ is then computed as

$$l = n - 1 + \log_2(w). \quad (18)$$

Note that the filter width is expressed in canonic texture coordinates with a range of $(u, v) \in [0, 1] \times [0, 1]$. A filter width of $w = 1$ means that we need to average over the whole texture. Therefore the mip-map level for $w = 1$ is $n - 1$.

Pseudo-code for trilinear filtering is given below. It is important to realize that the function for bilinear interpolation takes $(u, v)$ coordinates in the range $[0, 1] \times [0, 1]$ as parameters, which need to be converted to the appropriate texel coordinates before looking up the texture.

```c
if(l<0)
    return bilinear(0,u,v);
else if(l>=n-1)
    return texel(n-1,0,0);
else
    int i = Floor2Int(l);
    float d = l - i;
    return (1-d)*bilinear(i,u,v) + d*bilinear(i+1,u,v);
```