3
Visualização de *TetraQuads*

No capítulo anterior vimos como é definido uma malha de *TetraQuads*. Iremos agora descobrir como visualizar essa malha utilizando uma adaptação do algoritmo de *ray casting* para tetraedros contendo *TetraQuads*. Toda etapa de visualização é programada em GPU.

O *pipeline* de visualização possui três etapas. A primeira etapa equivale ao *vertex shader* que recebe um a um os tetraedros da malha com as quádricas correspondentes e realiza a transformação projetiva dos vértices. A etapa seguinte equivale ao *geometry shader* onde fazemos a subdivisão dos tetraedros em triângulos como explicaremos na seção 3.3. A última etapa acontece no *fragment shader* onde recebemos como entrada da etapa anterior pontos de interseção de raios que atravessam o tetraedro partindo do observador. Por último, encontramos a interseção do segmento formado por esses pontos com o *TetraQuad* e podemos definir o seu valor por interpolação linear das quádricas no segmento (figura 3.1).

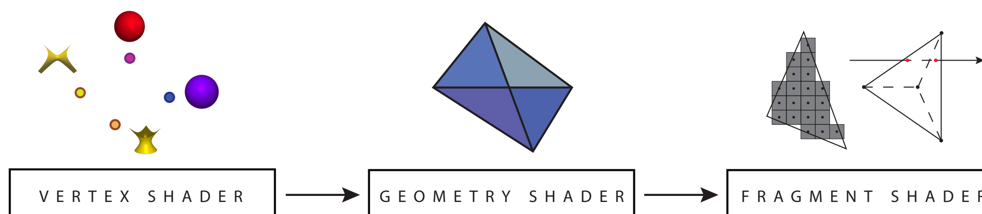

Figura 3.1: *Pipeline* de visualização.

3.1
Ray Casting

Também conhecido como traçado de raios, o algoritmo considera a tela como uma grade onde, a partir de cada célula (*pixels*), são lançados raios na direção de uma superfície que se deseja visualizar. Um raio r pode ser descrito

PUC-Rio - Certificação Digital Nº 1012858/CA

pela sequinte equação em t:

$$r(t) = (x(t), y(t), z(t)) = o + t\vec{v},$$

onde o ponto o é a posição da câmera e $\vec{v}$ é o vetor que a liga ao *pixel*.

Considere o exemplo clássico da aplicação do algoritmo para visualização de uma esfera como na figura 3.2. O passo seguinte ao lançamento de cada um dos raios é encontrar os valores de t que satisfazem a seguinte equação quadrática

$$x(t)^2 + y(t)^2 + z(t)^2 - c = 0.$$

Feito isso, escolhemos o valor que representa o ponto mais próximo do observador ou da câmera.

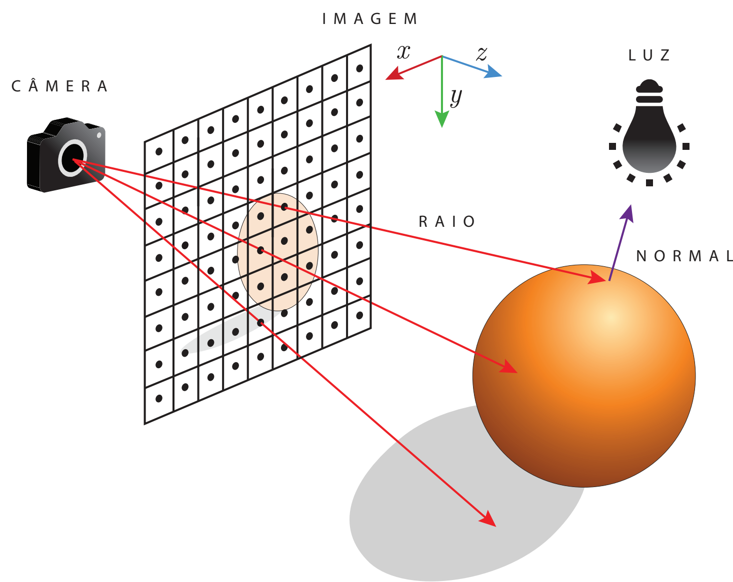

Figura 3.2: *Ray casting* de uma esfera.

PUC-Rio - Certificação Digital Nº 1012858/CA

Por fim, para cada raio que intersecta a superfície, determinamos a normal no ponto de interseção e com essas informação conseguimos determinar a cor do *pixel* utilizando algoritmos de iluminação como *Phong Shading* (17).

3.2
Ray Casting de *TetraQuads*

Para visualizar os *TetraQuads*, utilizamos o algorítimo de *ray casting* implementado em GPU semelhante ao que descrevemos para visualizar quádricas no capítulo anterior. Utilizamos o método de Newton (18) para achar as raízes da equação de interseção entre o raio $r(t)$ e a superfície no interior de cada

tetraedro. Inicialmente, para cada raio, precisamos achar os pontos de interseção com o tetraedro T (figura 3.3). Isso é feito para restringir o intervalo de procura de raízes no método de Newton já que estamos interessados apenas em pontos de interseção dentro do tetraedro. Ao ponto de interseção mais próximo do observador damos o nome de p_{in} e ao mais afastado o nome de p_{out}. Antes de aplicarmos o método de Newton, esse intervalo é restringido ainda mais através do método da Bisseção (18). Esses passos são necessários para garantir a convergência do algorítimo para a raiz mais próxima do observador.

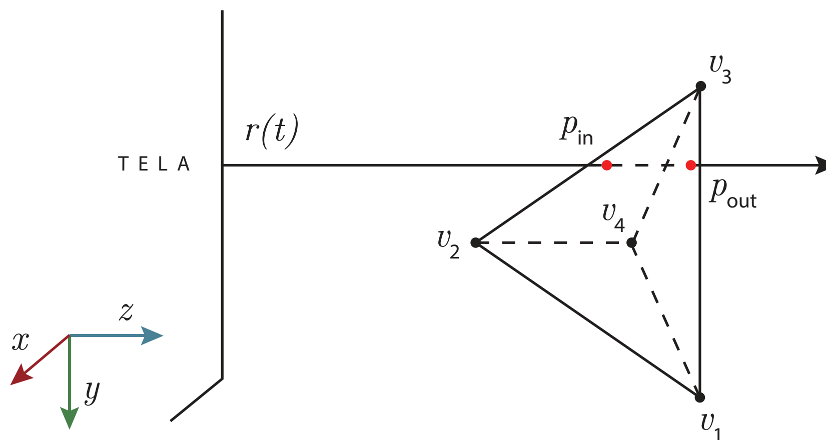

Figura 3.3: Encontrando os pontos de interseção entre o raio e o tetraedro.

3.3 Divisão dos Tetraedros

Encontrar os pontos de entrada e saída do raio no tetraedro é relativamente simples se considerarmos a projeção do tetraedro na tela, o que acontece num processo conhecido como tecelagem, implementado em transistores na GPU. Essa é a mesma abordagem adotada por Loop e Blinn (10) escolhida por ser natural para a placa gráfica, que enxerga os tetraedros como um conjunto de triângulos, a diferença é que fizemos a implementação no *shader* de processamento geométrico, inexistente na época.

Os casos em que um dos vértices é escondido na projeção do tetraedro por uma aresta ou outro vértice formando **um** ou **dois triângulos** são chamados de **degenerados** e são desconsiderados pela GPU. Dessa maneira, levamos em consideração apenas os dois casos não-degenerados. A figura 3.4 ilustra os quatro casos possíveis, observe que estamos falando dos triângulos formados pela projeção e não das faces originais do tetraedro.

O desafio inicial é definir em qual caso **não-degenerado** estamos, ou seja, se temos **três** ou **quatro triângulos**. Para isso, o primeiro passo é

PUC-Rio - Certificação Digital Nº 1012858/CA

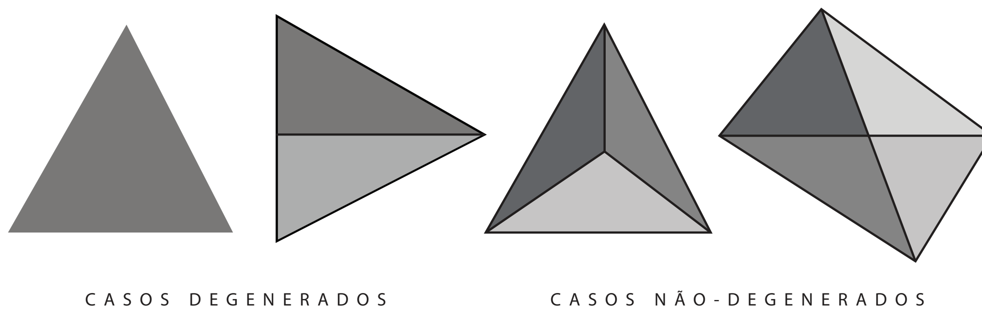

Figura 3.4: Casos de subdivisão do tetraedro.

ordenar os vértices do tetraedro projetado. Começamos pelo ponto mais abaixo e ordenamos os vértices seguintes no sentido anti-horário tomando como referência o vetor $\vec{v} = [1, 0]$. A figura 3.5, mostra a ordenação dos vértices para os dois casos considerados.

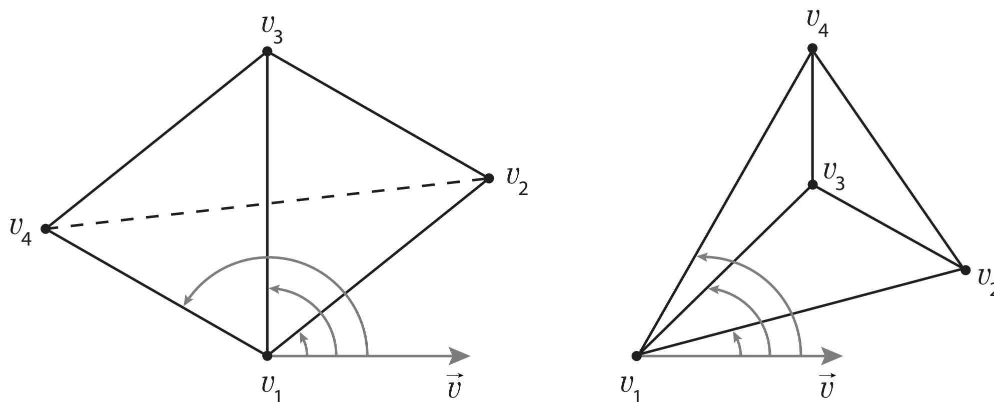

Figura 3.5: Ordenando os vértices do tetraedro.

PUC-Rio - Certificação Digital Nº 1012858/CA

Quatro triângulos. Feito isso, precisamos determinar o ponto onde os segmentos v_1v_3 e v_2v_4 se encontram, considerando apenas a projeção dos vértices na tela. Para isso, dados os segmentos bidimensionais $r_1(t) = tv_1 + (1-t)v_3$ e $r_2(s) = sv_2 + (1-s)v_4$, basta verificar se existem valores de $s \in (0, 1)$ e $t \in (0, 1)$ tais que $r_1(t) = r_2(s)$. Se esse fato for verdade temos o caso com quatro triângulos. O ponto de interseção mais próximo do observador chamamos de c_{in} e o mais afastado c_{out} (figura 3.6). Vamos considerar, por simplicidade, que o ponto mais próximo está no segmento r_1 e o mais afastado em r_2, fato que é fácil verificar bastando apenas substituir os valores de t e s nas equações

dos segmentos utilizando a coordenada z dos vértices originais e comparar os resultados.

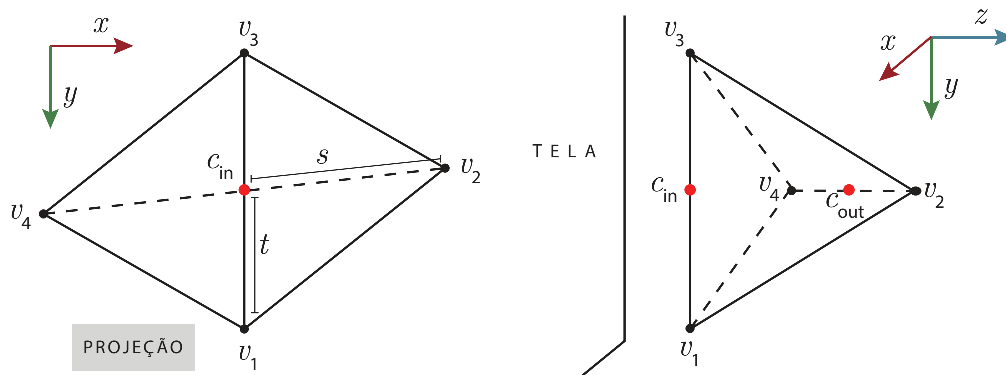

Figura 3.6: Projeção de T com quatro triângulos. Os vértices c_{in} e c_{out} são obtidos por interpolação linear nas arestas v_1v_3 e v_2v_4

A próxima etapa é emitir, ou seja, passar para a etapa de rasterização, os triângulos visíveis para o observador $c_{in}v_1v_2$, $c_{in}v_2v_3$, $c_{in}v_3v_4$ e $c_{in}v_4v_1$. Com cada um dos vértices passamos o ponto que primeiro intersecta o tetraedro, p_{in}, juntamente com a quádrica interpolada nesse ponto que chamamos de Q_{in} e o segundo a intersectar, p_{out}, com a quádrica Q_{out}. Para os vértices v_1, v_2, v_3 e v_4, obviamente, $p_{in} = p_{out}$ e seus valores são iguais aos próprios vértices assim como as quádricas $Q_{in} = Q_{out}$ que são passadas junto com cada um deles. Para o vértice c_{in} temos $p_{in} = c_{in}$, $p_{out} = c_{out}$ e as quádricas são obtidas por interpolação nos segmentos fazendo $Q_{in} = tQ_1 + (1-t)Q_3$ e $Q_{out} = sQ_2 + (1-s)Q_4$. Note que a cúbica do *TetraQuad* é obtida a partir da interpolação das quádricas, que por sua vez é feita em dois passos: interpolação em planos ortogonais ao raio (Q_{in} / Q_{out}) e depois ao longo do raio (no método de Newton do *fragment shader*).

Três triângulos. Como mencionamos no início, a GPU facilita o processo ao desconsiderar os casos degenerados. Se não houver interseção entre os segmentos, considerando a maneira como ordenamos os vértices, estamos no caso de projeção com três triângulos. Já sabemos que o vértice v_3 é o vértice central (figura 3.5), basta determinarmos agora se v_3 está mais próximo do observador ou se a sua projeção no triângulo $v_1v_2v_4$ tem a menor distância. Para isso, calculamos as coordenadas baricêntricas de v_3 relativas ao triângulo $v_1v_2v_4$ projetado no plano $z = 0$ (figura 3.7). Utilizamos esses valores para encontrar a projeção do vértice no triângulo original. Comparando a coordenada z de v_3

PUC-Rio - Certificação Digital Nº 1012858/CA

e do ponto projetado podemos definir c_{in}, o ponto mais próximo do observador, e c_{out}, o ponto mais distante.

Para determinar Q_{in} e Q_{out}, vamos considerar na nossa discussão que v_3 está mais próximo do observador como na figura 3.7. Nesse cenário, são emitidos os triângulos $c_{in}v_1v_2$, $c_{in}v_2v_4$ e $c_{in}v_4v_1$ com $p_{in} = p_{out}$ e $Q_{in} = Q_{out}$ para os vértices v_1, v_2 e v_4 como no primeiro caso. Já para o vértice c_{in} temos $p_{in} = c_{in}$, que é o próprio vértice v_3, logo $Q_{in} = Q_3$. O vértice c_{out} é a projeção de v_3 no triângulo $v_1v_2v_4$, logo $p_{out} = c_{out} = b_1v_1 + b_2v_2 + b_3v_4$ e $Q_{out} = b_1Q_1+b_2Q_2+b_3Q_4$. Os valores b_1, b_2 e b_3 são as coordenadas baricêntricas do vértice v_3 obtidas durante o processo.

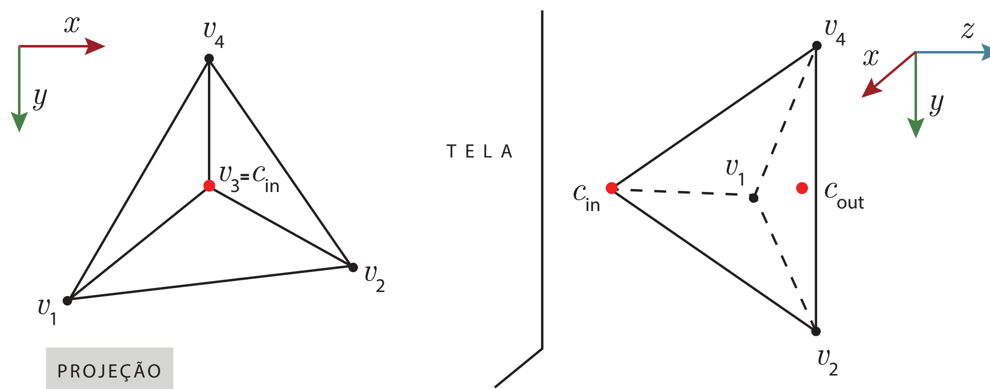

Figura 3.7: Projeção de T com três triângulos. O vértice $c_{out} = b_1v_1+b_2v_2+b_3v_4$ é obtido projetando o vértice v_3 no triângulo $v_1v_2v_4$.

3.4 Interpolação

Como acabamos de ver, junto com os vértices do tetraedro são passadas as quádricas associadas a cada um deles. As quádricas dos pontos centrais são definidas por interpolação. No primeiro caso é utilizada interpolação linear nas arestas e no segundo interpolação baricêntrica na face do tetraedro onde o ponto foi projetado. Para cada um dos pontos encontrados, o mais próximo chamamos de p_{in} e o mais distante de p_{out} e às quádricas associadas a cada um deles damos o nome de Q_{in} e Q_{out}.

Os triângulos construídos são passados para a próxima etapa do pipeline de renderização. Essa etapa constitui o processo de converter em *pixels* os triângulos obtidos durante a etapa de tecelagem. A figura 3.8 ilustra o processo de rasterização de uma das faces do tetraedro.

PUC-Rio - Certificação Digital Nº 1012858/CA

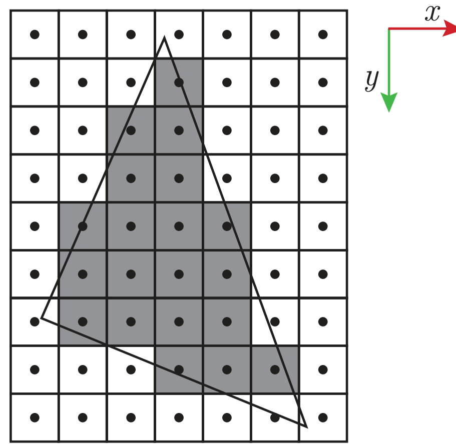

Figura 3.8: Processo de *rasterização* de uma das faces do tetraedro.

No processo de rasterização, todos os outros pontos de interseção, p_{in} e p_{out}, e as quádricas associadas a eles são obtidos com correção de perspectiva pela própria placa gráfica por interpolação baricêntrica nos triângulos emitidos. A partir disso, para cada *pixel*, percorremos o segmento formado pelos pontos p_{in} e p_{out} e com o método de Newton encontramos o ponto de interseção do raio com a superfície de nível 0 mais próximo do observador. Um ponto qualquer ao logo do raio que atravessa o tetraedro é descrito pela seguinte equação em t:

$$p(t) = tp_{in} + (1-t)p_{out}, \tag{3-1}$$

onde t é um parâmetro variando no intervalo $[0, 1]$.

E, equivalentemente, a quádrica ao longo do raio é definida por

$$Q(t) = tQ_{in} + (1-t)Q_{out}. \tag{3-2}$$

Definir se o o ponto onde o raio intersecta o *TetraQuad* equivale a achar o t que satisfaça a equação cúbica abaixo:

$$F(t) = C(p(t)) = p(t) \cdot Q(t) \cdot p(t)^{\top} = 0. \tag{3-3}$$

Utilizamos o método de Newton como mencionamos na seção 3.2 para achar o valor de t equivalente ao ponto mais próximo do observador. É fácil mostrar que estamos diante de uma expressão cúbica, basta substituir o valor das equações

PUC-Rio - Certificação Digital Nº 1012858/CA

(3-1) e (3-2). Dessa maneira:

$$\begin{aligned}
F(t) &= (tp_{in} + (1-t)p_{out}) \cdot (tQ_{in} + (1-t)Q_{out}) \cdot (tp_{in} + (1-t)p_{out})^\top \\
&= t^3(p_{in}Q_{in}p_{in}^\top - 2p_{in}Q_{in}p_{out}^\top - p_{in}Q_{out}p_{in}^\top + 2p_{in}Q_{out}p_{out}^\top + p_{out}Q_{in}p_{out}^\top \\
&- p_{out}Q_{out}p_{out}^\top) + t^2(p_{in}Q_{in}p_{in}^\top + p_{in}Q_{out}p_{in}^\top - 3p_{in}Q_{out}p_{out}^\top - p_{out}Q_{in}p_{out}^\top \\
&+ 3p_{out}Q_{out}p_{out}^\top) + t(p_{in}Q_{in}p_{out}^\top + p_{in}Q_{out}p_{out}^\top - 3p_{out}Q_{out}p_{out}^\top) \\
&+ p_{out}Q_{out}p_{out}^\top.
\end{aligned}$$

Agora que sabemos encontrar o ponto de interseção, precisamos encontrar o valor do vetor normal para definir sua cor (a última etapa do algorítimo de *ray casting*). Melhores detalhes a respeito desse processo podem ser vistos na seção a seguir.

3.5 Normais

A expressão da normal da quádrica Q em um dado ponto p, como foi mostrado no trabalho de Toledo (8), é dada pela equação abaixo:

$$\vec{n} = Q' \cdot p, \tag{3-4}$$

onde Q' é a submatriz $[3 \times 3]$ extraída do lado superior esquerdo de Q.

Cada valor de t que satisfaça a equação (3-3) equivale a um ponto de interseção p_t e uma quádrica Q_t, substituindo seu valor nas equações (3-1) e (3-2). Dessa forma, podemos achar o vetor normal no ponto de interseção utilizando a expressão da normal da quádrica:

$$\vec{n}_t = Q'_t \cdot p_t. \tag{3-5}$$

É fácil calcular a normal do *TetraQuad* a partir da equação (2-8), mas precisaríamos passar os quatro vértices do tetraedro para o fragmento durante o processo de rasterização. Um custo que mostrou-se desnecessário já que o vetor $\vec{n}_t$ é uma boa aproximação para a normal do *TetraQuad* no ponto p_t. Além disso, a descontinuidade que existe no campo de normais formado pelos vetores $\vec{n}_t$ não seria corrigida calculando a normal do *TetraQuad* que também possui uma descontinuidade nesse caso como mencionamos na seção 2.4.

Agora que definimos o que é o *TetraQuad* e que sabemos como visualizar esses objetos veremos como modelar um sistema para reconstruir superfícies utilizando esses tetraedros a partir de um conjunto de pontos de entrada. Existem vários parâmetros a serem levados em consideração para obter uma boa aproximação. Entretanto, já podemos adiantar que defini-los continua sendo um desafio como pode ser observado na figura 3.9 que mostra problemas

PUC-Rio - Certificação Digital Nº 1012858/CA

ainda não resolvidos na reconstrução do modelo *Bunny*. Iremos discutir melhor esse tipo de problema no próximo capítulo.

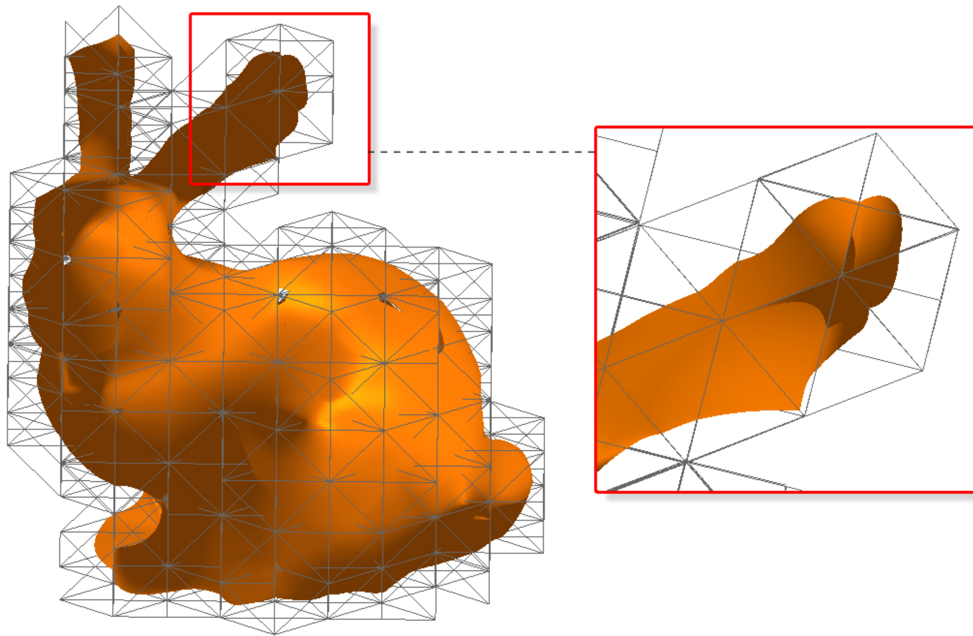

Figura 3.9: Neste exemplo podemos perceber problemas na reconstrução por *TetraQuads* na orelha do *Bunny* que tem uma geometria um pouco mais delicada. Uma melhor amostragem dos pontos na região, alterações na malha ou ajustes nos parâmetros do algoritmo de mínimos quadrados podem amenizar ou até mesmo resolver esse tipo de problema.

PUC-Rio - Certificação Digital Nº 1012858/CA