3
Sistema Estudado

Neste capítulo descrevemos o sistema estudado e posteriormente implementado para a captura de modelos computacionais e os conceitos teóricos envolvidos. Cada ponto é aprofundado nas seções correspondentes, divididas em calibração de câmeras, luz estruturada codificada utilizada, captura e processamento de imagens, triangulação e modelo final.

3.1.
Calibração de Câmera

O primeiro passo do sistema estudado consiste em calibrar as duas câmeras. Calibrar uma câmera é determinar os valores dos parâmetros extrínsecos e intrínsecos da mesma. Em outras palavras, a calibração consiste em determinar a posição e orientação da câmera em relação a uma referência fixa e obter suas características óptica, geométrica e digital.

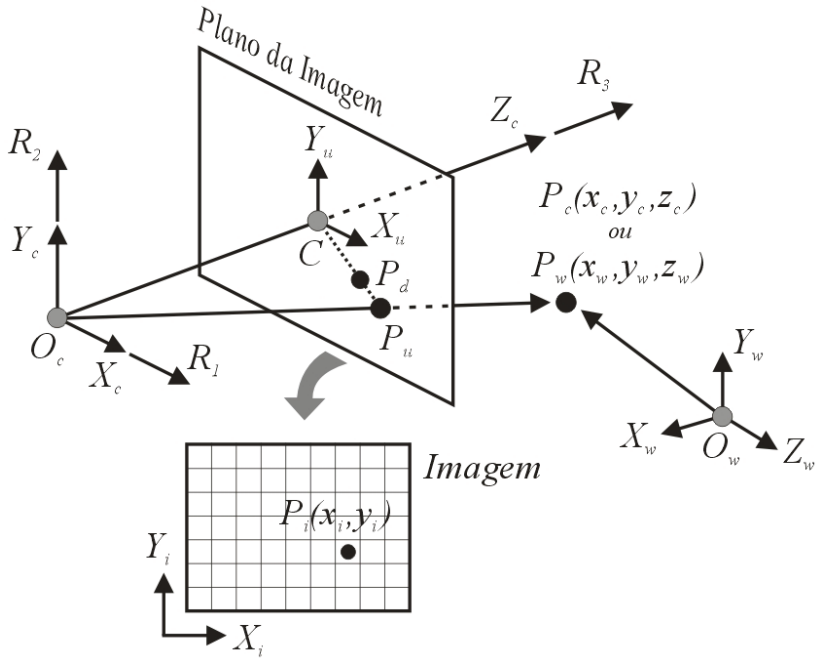

Figura 10 – Modelos de Câmera de Tsai.

Entre os diversos métodos existentes para calibração de câmera, utilizou-se o método proposto por Tsai em [17] e [18] nesta tese. O modelo de câmera de Tsai é baseado no modelo de projeção perspectiva de câmera completa, com lentes, e possui um conjunto de parâmetros extrínsecos e outro de parâmetros intrínsecos. Cada conjunto de parâmetros pode ser estimado separadamente. Os parâmetros utilizados estão descritos nesta seção.

A figura 10 ilustra o modelo utilizado. O sistema de coordenadas do mundo é definido por (X_w, Y_w, Z_w), sendo O_w a origem do mundo. O sistema de coordenadas em relação à câmera é dado por (X_c, Y_c, Z_c), sendo O_c a origem deste sistema. O primeiro passo do algoritmo é determinar a posição e orientação da câmera em relação a uma referência fixa. Para isso utilizamos um conjunto de parâmetros que identificam univocamente a transformação entre a posição e orientação da câmera e a posição e orientação de referência. Estes são os parâmetros extrínsecos da câmera.

Os parâmetros extrínsecos são dados por:

- Um vetor de translação T que descreve a posição da origem O_c em relação à origem de referência O_w.
- Uma matriz ortogonal de rotação R que leva os eixos de referência aos eixos correspondentes da câmera.

Seja P um ponto qualquer no mundo. Suas coordenadas são dadas por (x_w, y_w, z_w). No sistema da câmera, suas coordenadas são dadas por (x_c, y_c, z_c). A relação entre estas coordenadas é descrita por:

$$P_c = R(P_w - T) \tag{3.1}$$

A imagem abaixo ilustra esta transformação.

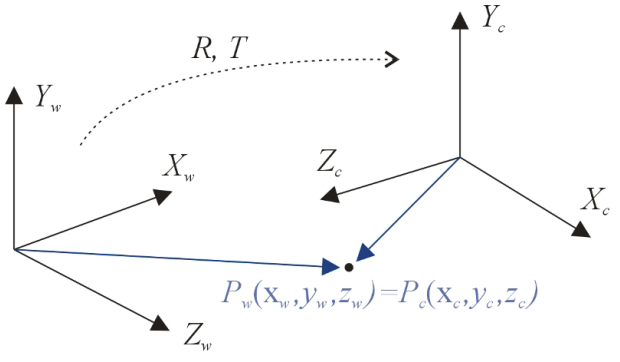

Figura 11 – Relação entre coordenadas do mundo e da câmera.

O ponto O_c também é o centro de projeção da câmera. O eixo Z_c coincide com o eixo óptico e corta o plano da imagem no ponto C, chamado de centro da imagem. O segundo passo do algoritmo é achar o ponto correspondente à projeção perspectiva de P no plano da imagem, dado por P_u.

Podemos caracterizar os parâmetros intrínsecos como o conjunto de parâmetros necessários para definir as características ópticas, geométricas e digitais de uma câmera. Para o modelo perspectivo de câmera são necessários três subconjuntos de parâmetros intrínsecos que especificam:

- A projeção perspectiva.
- A distorção geométrica introduzida pelo sistema de lentes da câmera.
- A transformação entre coordenadas no plano da imagem e coordenadas na imagem capturada.

Para a projeção perspectiva, o único parâmetro utilizado é a distância focal f, correspondente à distância entre os pontos O_c e C. As coordenadas do ponto P_u são dadas por (x_u, y_u) no sistema de coordenadas definido por (X_u, Y_u) e tendo como origem o ponto C. Logo temos as seguintes relações:

$$x_u = f \frac{x_c}{z_c} \tag{3.2}$$

$$y_u = f \frac{y_c}{z_c} \tag{3.3}$$

Este seria o caso considerado se não houvesse nenhuma distorção óptica. P_d é o ponto correspondente à projeção de P, levando em conta distorções radiais introduzidas pelas lentes. O terceiro passo é transformar as coordenadas no plano da imagem para as coordenadas correspondentes com distorções ópticas.

Distorções radiais são deslocamentos radiais de pontos no plano da imagem. Quanto mais afastado o ponto está do centro da imagem, maior é seu deslocamento. Distorções radiais são normalmente modeladas por:

$$x_u = x_d\left(1 + k_1 r^2 + k_2 r^4\right) \tag{3.4}$$

$$y_u = y_d\left(1 + k_1 r^2 + k_2 r^4\right) \tag{3.5}$$

O quadrado da distância é dado por:

$$r^2 = x_d^2 + y_d^2 \tag{3.6}$$

As coordenadas no plano da imagem de um ponto com distorções radiais são dadas por (x_d, y_d) e (x_u, y_u) são suas coordenadas já livres de distorções da lente, ambas no mesmo sistema de coordenadas. Os dois parâmetros intrínsecos são k_1 e k_2 que caracterizam a distorção geométrica introduzida pelo sistema de lentes da câmera. No modelo original de Tsai, k_2 é descartado por ser usualmente muito baixo e tem seu valor igual a zero.

O último passo corresponde à transformação de coordenadas no plano da imagem para coordenadas na imagem capturada. Para isso temos as seguintes equações:

$$x_d = (x_i - o_x)s_x \tag{3.7}$$

$$y_d = (y_i - o_y)s_y \tag{3.8}$$

As coordenadas do ponto P_i na imagem é dado por (x_i, y_i) em unidades de pixel e as coordenadas do ponto correspondente P_d no plano da imagem é dado por (x_d, y_d) em metros. O par de parâmetros (o_x, o_y) definem o centro da imagem, correspondente ao ponto C no plano da imagem, e é utilizado como o centro do sistema de coordenadas da imagem capturada representado por (X_i, Y_i). Por último, o par (s_x, s_y) definem o tamanho efetivo de um pixel, em metros por pixel, nas direções horizontais e verticais respectivamente. Estes são os fatores de escala. No modelo original de Tsai, s_x e s_y são considerados iguais e modelados por um único parâmetro s.

Podemos resumir o algoritmo nos quatro passos apresentados, como mostra a figura 12 abaixo. Dado um ponto em coordenadas do mundo, o algoritmo retorna suas coordenadas na imagem final. No total temos nove parâmetros que devem ser estimados para cada câmera calibrada.

Se considerarmos uma câmera ideal, a qual não possui distorção radial e o centro da imagem capturada coincide com o centro do plano da imagem, o modelo de câmera de Tsai pode ser simplificado a um modelo de câmera "pinhole", encontrada na literatura. Se incluirmos o fator de escala s no valor da distância focal f, podemos simplificar o algoritmo de Tsai descartando os passos três e quatro. Embora estas simplificações sejam utilizadas por várias aplicações, não existe câmera perfeita. Logo há um comprometimento na precisão da calibração, que varia com a câmera utilizada.

Passo 1 Transformação para Coordenadas da Câmera		
Entrada	Parâmetros	Saída
$P_w(x_w, y_w, z_w)$	R, T	$P_c(x_c, y_c, z_c)$

Passo 2 Projeção Perspectiva		
Entrada	Parâmetros	Saída
$P_c(x_c, y_c, z_c)$	f	$P_u(x_u, y_u)$

Passo 3 Distorção Radial		
Entrada	Parâmetros	Saída
$P_u(x_u, y_u)$	k_1, k_2	$P_d(x_d, y_d)$

Passo 4 Transformação para Coordenadas da Imagem Capturada		
Entrada	Parâmetros	Saída
$P_d(x_d, y_d)$	o_x, o_y, s_x, s_y	$P_i(x_i, y_i)$

Figura 12 – Os quatro passos do algoritmo de Tsai.

Como foi citado, os parâmetros extrínsecos e intrínsecos podem ser estimados separadamente. Com os valores dos parâmetros intrínsecos da câmera previamente calculados, podemos calcular a partir de uma imagem capturada uma segunda imagem livre de distorções radiais. A nova imagem é criada utilizando-se as equações (3.4) a (3.8). O fator de escala *s* é incluído no valor da distância focal *f* e possui valor igual a um nas equações em que é utilizada. Com isso simplificamos novamente o modelo utilizado a um modelo de câmera "pinhole", com o fator de escala incluído na distância focal. A diferença é que os parâmetros intrínsecos são previamente calibrados e utiliza-se a imagem livre de distorções radiais na calibração dos parâmetros extrínsecos. Novamente podemos simplificar o algoritmo de Tsai descartando os passos três e quatro. Para estimar os parâmetros intrínsecos, utilizou-se o algoritmo proposto por Zhang em [19].

Para determinar os valores dos parâmetros extrínsecos nos passos um e dois, utilizou-se o algoritmo de Tsai coplanar, onde todos os pontos do padrão de calibração, usados para encontrar os parâmetros, estão no plano $z_w = 0$.

Os parâmetros extrínsecos podem ser escritos na forma matricial:

$$R = \begin{bmatrix} \vec{r}_1 \\ \vec{r}_2 \\ \vec{r}_3 \end{bmatrix} = \begin{bmatrix} r_{1x} & r_{1y} & r_{1z} \\ r_{2x} & r_{2y} & r_{2z} \\ r_{3x} & r_{3y} & r_{3z} \end{bmatrix} \quad T = \begin{bmatrix} t_x \\ t_y \\ t_z \end{bmatrix}$$

Os vetores $\vec{r}_1$, $\vec{r}_2$ e $\vec{r}_3$ são ortonormais entre si e estão ilustrados na figura do modelo. Estas matrizes podem ser vistas como uma única matriz $[R \mid T]$.

$$[R \mid T] = \begin{bmatrix} \vec{r}_1 & t_x \\ \vec{r}_2 & t_y \\ \vec{r}_3 & t_z \end{bmatrix} = \begin{bmatrix} r_{1x} & r_{1y} & r_{1z} & t_x \\ r_{2x} & r_{2y} & r_{2z} & t_y \\ r_{3x} & r_{3y} & r_{3z} & t_z \end{bmatrix} \tag{3.9}$$

As equações (3.2), (3.3) e (3.9) podem ser compatibilizados de forma a produzir uma transformação direta do ponto (x_w, y_w, z_w) no mundo para o ponto (x_i, y_i) na imagem, como é feito em [20]. Devemos lembrar que, segundo as simplificações adotadas, a imagem utilizada está livre de distorções radiais e que o fator de escala *s* está incluído na distância focal *f*.

$$\begin{bmatrix} x_i s \\ y_i s \\ s \end{bmatrix} = \begin{bmatrix} f\vec{r}_1 & f\,t_x \\ f\vec{r}_2 & f\,t_y \\ \vec{r}_3 & t_z \end{bmatrix} \begin{bmatrix} x_w \\ y_w \\ z_w \\ 1 \end{bmatrix} \tag{3.10}$$

As coordenadas (x_i, y_i, s) são homogêneas. Multiplicando (3.9) obtemos duas equações:

$$x_i = f \frac{r_{1x}x_w + r_{1y}y_w + r_{1z}z_w + t_x}{r_{3x}x_w + r_{3y}y_w + r_{3z}z_w + t_z} \tag{3.11}$$

$$y_i = f \frac{r_{2x}x_w + r_{2y}y_w + r_{2z}z_w + t_y}{r_{3x}x_w + r_{3y}y_w + r_{3z}z_w + t_z} \tag{3.12}$$

Como estamos utilizando o método coplanar, podemos escrever $z_w = 0$ nas equações (3.11) e (3.12).

$$x_i = f \frac{r_{1x}x_w + r_{1y}y_w + t_x}{r_{3x}x_w + r_{3y}y_w + t_z} \tag{3.13}$$

$$y_i = f \frac{r_{2x}x_w + r_{2y}y_w + t_y}{r_{3x}x_w + r_{3y}y_w + t_z} \tag{3.14}$$

Dividindo (3.13) por (3.14) e em seguida o numerador e o denominador da razão direta por t_y obtemos:

$$x_i = \frac{r_{1x}}{t_y} x_w y_i + \frac{r_{1y}}{t_y} y_w y_i + \frac{t_x}{t_y} y_i - \frac{r_{2x}}{t_y} x_w x_i - \frac{r_{2y}}{t_y} y_w x_i \tag{3.15}$$

Podemos obter um sistema linear, com cada linha correspondendo a um ponto amostrado. Cada ponto possui um índice *k* diferente.

$$x_{i_k} = \frac{r_{1x}}{t_y} x_{w_k} y_{i_k} + \frac{r_{1y}}{t_y} y_{w_k} y_{i_k} - \frac{r_{2x}}{t_y} x_{w_k} x_{i_k} - \frac{r_{2y}}{t_y} y_{w_k} x_{i_k} + \frac{t_x}{t_y} y_{i_k} \tag{3.16}$$

Com isso obtemos o sistema linear $A\,u = b$, onde A é uma matriz $n \times 5$, cada linha A_k é dada por $\left(x_{w_k} y_{i_k}, y_{w_k} y_{i_k}, -x_{w_k} x_{i_k}, -y_{w_k} x_{i_k}, y_{i_k}\right)$, cada elemento do vetor *b* é dado por x_{i_k} e o vetor *u* é dado por:

$$\left[U_1\ U_2\ U_3\ U_4\ U_5\right] = \left[\frac{r_{1x}}{t_y} \quad \frac{r_{1y}}{t_y} \quad \frac{r_{2x}}{t_y} \quad \frac{r_{2y}}{t_y} \quad \frac{t_x}{t_y}\right] \tag{3.17}$$

Lembrando que $\vec{r}_1$, $\vec{r}_2$ e $\vec{r}_3$ são ortonormais e definindo $\alpha = \frac{r_{1z}}{t_y}$ e $\beta = \frac{r_{2z}}{t_y}$, temos que:

$$\begin{cases} \alpha\beta = -U_1 U_3 - U_2 U_4 \\ \alpha^2 + U_1^2 + U_2^2 = \beta^2 + U_3^2 + U_4^2 \end{cases} \tag{3.18}$$

Resolvendo o sistema acima, calcula-se t_y a partir de:

$$t_y^2 = \frac{U - \sqrt{U^2 - 4\left(U_1 U_4 - U_2 U_3\right)^2}}{2\left(U_1 U_4 - U_2 U_3\right)^2} \tag{3.19}$$

$$U = U_1^2 + U_2^2 + U_3^2 + U_4^2$$

Com estes valores definidos e utilizando a equação (3.17) determina-se os valores de r_{1x}, r_{1y}, r_{2x}, r_{2y} e t_x. Como os vetores $\vec{r}_1$ e $\vec{r}_2$ são normalizados, obtemos os valores de r_{1z} e r_{2z}. O vetor $\vec{r}_3$ pode ser calculado, já que $\vec{r}_1$, $\vec{r}_2$ e $\vec{r}_3$ são ortonormais. Finalmente, utilizando os valores já encontrados e as equações (3.13) e (3.14), podemos calcular f e t_z.

As próximas duas subseções apresentam dois padrões de calibração, utilizados na obtenção de pontos para a calibração, e como extrair seus pontos característicos das imagens capturadas. Ambos os padrões são coplanares.

3.1.1. Padrão com Círculos

Entre os vários padrões utilizados em calibração de câmeras, este padrão coplanar com círculos é um dos mais fáceis de ser implementado e utilizado. De uma maneira geral, o padrão é constituído por vários círculos desenhados em um mesmo plano e afastados um do outro. Podemos utilizar uma folha rígida de cartolina para construir o padrão. Círculos formam padrões facilmente identificáveis na imagem que podem ser consideradas elipses. Alternativamente podemos utilizar esferas posicionadas também em um mesmo plano e distribuídas de maneira análoga. Esferas formam círculos na imagem facilmente identificáveis, porém elas podem gerar sombras na cena capturada dificultando a calibração. Em ambos os casos devemos extrair as elipses da imagem capturada e retornar o centro de cada uma para a calibração. A figura 13 ilustra o processo de obtenção de pontos do padrão.

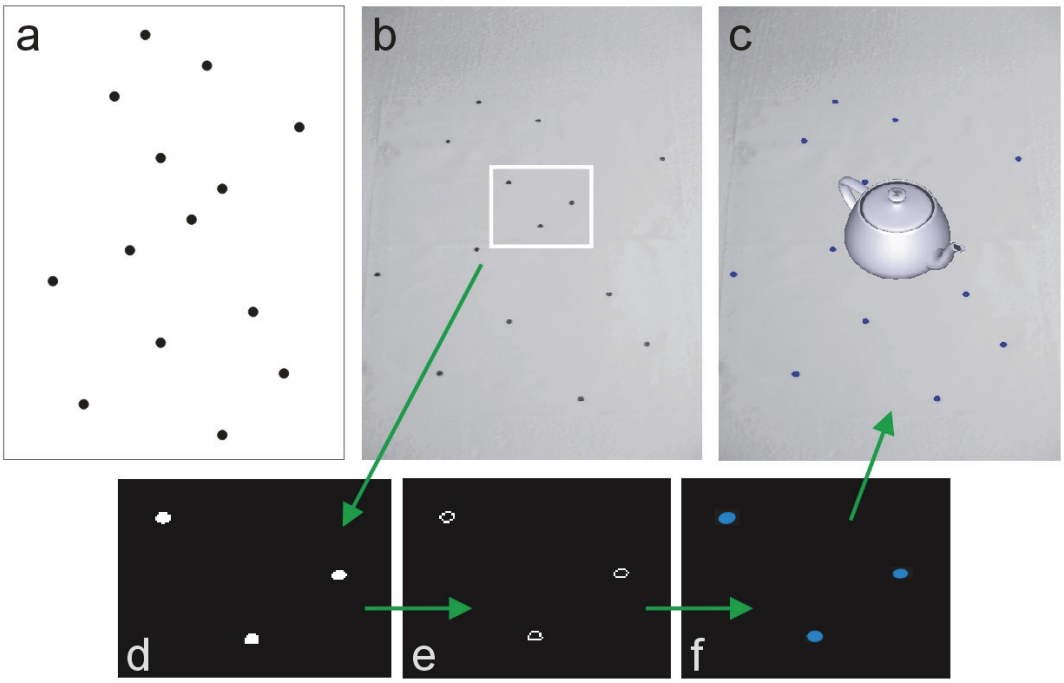

Figura 13 – Processo de obtenção de pontos do padrão com círculos: (a) desenho do padrão; (b) imagem capturada do padrão; (c) resultado final; (d) imagem binária; (e) bordas das componentes conexas encontradas; (f) elipses encontradas.

A primeira figura 13a mostra o desenho do padrão coplanar. Os círculos estão dispostos sobre o padrão de maneira que cada um não possui nenhum vizinho na direção horizontal. Esta disposição permite identificar cada círculo na

imagem capturada, mostrada na figura 13b já em tons de cinza. Basta seguir a ordem dos círculos na direção vertical. Para separar os círculos do fundo, utiliza-se um filtro "threshold" binário, dado por:

$$dst(x,y) = \begin{cases} 0, & se\ src(x,y) > threshold \\ 1, & caso\ contrário \end{cases} \tag{3.20}$$

onde *src* é a imagem de origem, *dst* é a imagem de destino binária e de mesmo tamanho e *threshold* é um valor limite. A figura 13d mostra o resultado deste filtro aplicado à imagem 13b. O valor de *threshold* utilizado comumente é 128 em uma escala de tons de cinza de 8 bits.

O próximo passo é identificar as bordas das componentes conexas da imagem binária. Componente conexa é uma região formada por pixels conectados uns aos outros. A relação entre os pixels conectados é dada pela conectividade escolhida, que estabelece condições relativas à cor do pixel e adjacência.

Para a primeira condição de conectividade, os pixels vizinhos devem possuir um mesmo valor de intensidade do conjunto V. No caso de imagem binária, $V=\{1\}$. Para o critério de adjacência, deve-se introduzir primeiro o conceito de vizinhança. Seja p um pixel na imagem dado pelas suas coordenadas (x,y), o conjunto de pixels chamado de 4-vizinhança é dado por:

$$N_4(p) = \{(x+1,y),(x-1,y),(x,y+1),(x,y-1)\}$$

De forma análoga, sua 8-vizinhança é dado por:

$$N_8(p) = N_4(p) \cup \{(x+1,y+1),(x-1,y+1),(x-1,y-1),(x+1,y-1)\}$$

Então podemos dizer que dois pixels p e q são 4-conectados se q pertence ao conjunto $N_4(p)$. Da mesma forma, p e q são 8-conectados se q pertence ao conjunto $N_8(p)$. A figura 14 ilustra duas componentes conexas, cujos pixels estão 4-conectados em uma imagem binária de tamanho 8×8. Pixels vizinhos estão conectados por segmentos de reta.

No algoritmo proposto, depois de obtidas as componentes conexas da imagem binária utilizando a conectividade com oito vizinhos, deve-se retornar apenas a borda de cada componente conexa para o próximo passo. Isso é feito facilmente descartando todos os pixels que possui oito elementos no seu conjunto $N_8(p)$ em cada componente conexa. O resultado desta operação pode ser visto na figura 13e.

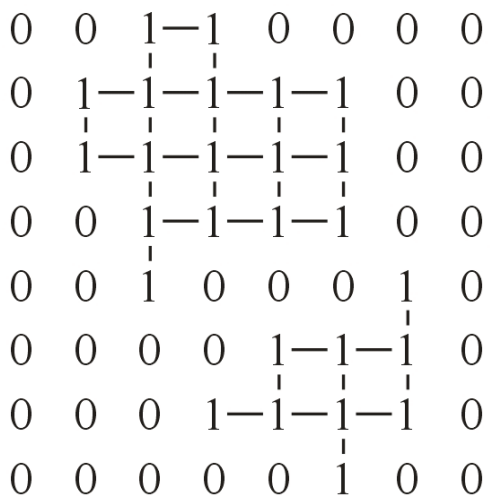

Figura 14 – Componentes conexas em uma imagem binária.

Cada borda de uma componente conexa representa o contorno de um círculo capturado e possui a forma de uma elipse. O último passo se resume a encontrar o centro de cada uma das elipses. Nas condições de uso do padrão, onde a câmera a ser calibrada é posicionada em frente ao padrão, o centro da elipse pode ser considerado o centro do círculo, sem perda de precisão. Uma maneira fácil de se achar o centro da elipse é achar a posição média dos pontos da borda da sua componente conexa. Com isso obtemos o centro de elipse com grande precisão. A figura 13f mostra as elipses achadas para cada componente conexa e a figura 13c mostra o resultado da calibração, com um objeto virtual inserido na imagem capturada. No caso de utilizarmos o padrão com esferas distribuídas no mesmo plano, obteremos círculos na imagem, que são casos particulares de elipses. Podemos empregar um dos métodos dados em [21] para achar o centro de cada um dos círculos.

O maior problema de se utilizar este padrão é posicioná-lo de tal maneira que apenas os círculos sejam capturados na imagem. Também há restrição na orientação do padrão na imagem capturada. No fim, o posicionamento fica bem limitado e objetos no fundo da imagem podem impedir a calibração. Para resolver estes problemas práticos, desenvolveu-se um outro padrão mais robusto, mostrado a seguir.

3.1.2.
Padrão com Vértices

Assim como o outro padrão apresentado, este padrão é coplanar e seu formato é dado pela figura 15. Ele possui doze vértices numerados e foi originalmente utilizado em uma outra aplicação. Porém seu uso pode ser facilmente adaptado para diversas aplicações. Estamos interessados em encontrar e retornar a posição de todos os vértices do padrão na imagem capturada. Estes são os pontos utilizados na calibração.

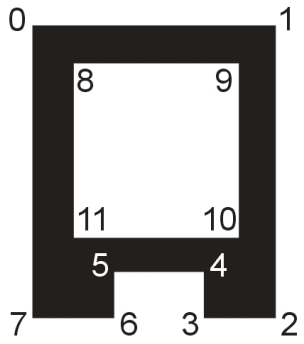

Figura 15 – Padrão com vértices.

Dado uma imagem capturada, o primeiro passo é aplicar o filtro "threshold" binário e depois achar as bordas de todas as componentes conexas na imagem, da mesma maneira de como foi feito no padrão anterior. O resultado desta operação de segmentação são várias bordas de componentes conexas, como está ilustrado na figura 16b. Observe que duas bordas de componentes conexas, entre várias encontradas, correspondem ao padrão de calibração, que iremos chamar de borda exterior e borda interior do padrão. Componentes conexas muito pequenas podem ser descartadas.

Para cada borda encontrada podemos organizar os seus pixels de maneira que eles formem uma lista circular, onde cada ponto é vizinho de seus pontos anterior e posterior na lista. O primeiro ponto também é vizinho do último ponto. Podemos obter mais de uma lista para cada borda, pois pode haver buracos nas componentes conexas, como é o caso do padrão. Desejamos encontrar as listas circulares correspondentes ao padrão de calibração e depois encontrar os pixels correspondente aos vértices nas listas circulares encontradas. Uma maneira de se resolver este problema é identificar a borda exterior. Encontrada a borda exterior,

fica fácil encontrar a borda interior, pois esta é dada pela outra lista circular da mesma componente conexa. As bordas candidatas à borda exterior devem possuir oito vértices.

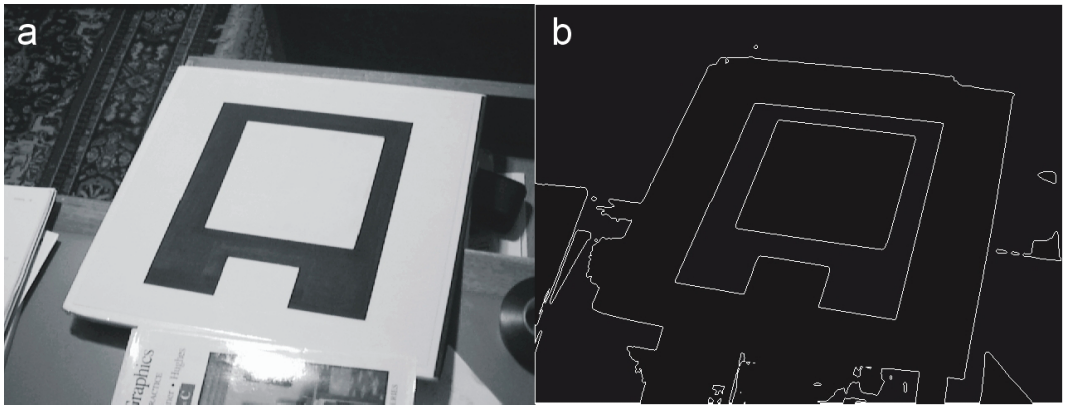

Figura 16 – Bordas das componentes conexas da imagem capturada: (a) imagem capturada do padrão; (b) bordas obtidas.

Para identificar qual pixel melhor corresponde a um determinado vértice, utilizamos uma heurística que se baseia na vizinhança de cada pixel. Dado um pixel na posição *m* em uma lista circular de pixels de tamanho *n*, sua posição na imagem é dada por (x_m, y_m), com $1 \le m \le n$. A distância do pixel na posição *m* da lista a um pixel na posição p é dado pelo vetor:

$$\overrightarrow{d_{m\,p}} = (x_p - x_m, y_p - y_m) \tag{3.21}$$

Considerando os *2k* pixels vizinhos mais próximos na lista do pixel de posição *m*, teríamos os pixels de índice variando de $m-k$ até $m+k$. Então para cada pixel na lista calcula-se o vetor resultante da soma da distância entre este pixel e seus *2k* pixels mais próximos na lista, dado por:

$$\overrightarrow{r_m} = \sum_{i=1}^{k} \overrightarrow{d_{m\,m+i}} + \overrightarrow{d_{m\,m-i}} \tag{3.22}$$

A figura 17 ilustra dois casos onde a resultante da soma das distâncias entre um pixel na posição *m* da lista e seus *2k* vizinhos é calculada. Cada pixel está representado na sua posição na imagem e seu número dá sua posição na lista a que pertence. Os vetores das distâncias entre o pixel central e seus vizinhos estão representados pelos vetores em preto e a resultante da soma de todos eles é dado por $\overrightarrow{r_m}$. Podemos observar que pixels correspondente a vértices possuem um vetor resultante de comprimento maior que seus vizinhos, enquanto que pixels no meio

de uma aresta possuem resultante de comprimento próximo de zero. Logo um pixel é considerado um vértice se:

1. Vetor resultante possui módulo maior que um valor de corte dado por $r_{\min}$.

$$\left|\overrightarrow{r_m}\right| \geq r_{\min}$$

2. O módulo do vetor resultante é maior ou igual que todos os módulos dos vetores resultantes dos *2k* pixels vizinhos.

$$\left|\overrightarrow{r_m}\right| \geq \left|\overrightarrow{r_p}\right|, \quad m-k \leq p \leq m+k$$

Como critério de desempate entre dois pixels vizinhos com o mesmo valor para o módulo da resultante, escolhemos sempre o primeiro na lista. A dificuldade deste método está na escolha dos valores de *k* e $r_{\min}$.

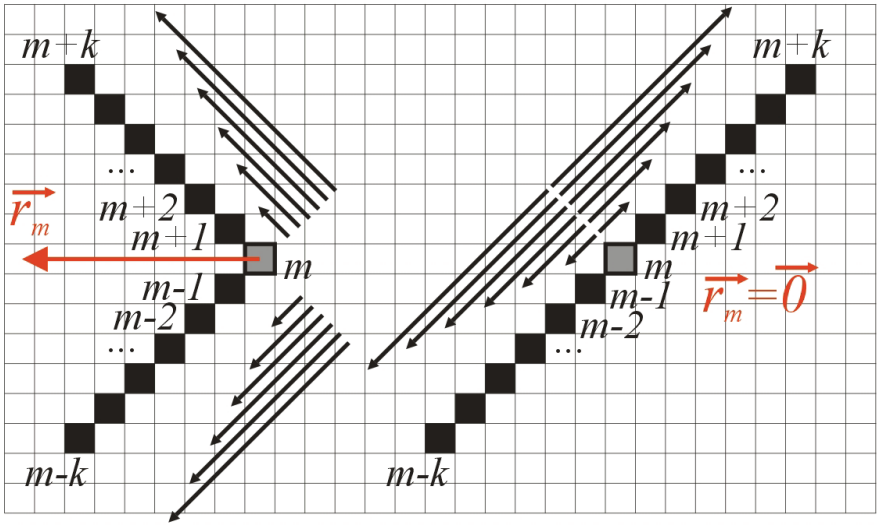

Figura 17 – Vetores resultantes da soma das distâncias de um pixel aos seus vizinhos.

Então utilizando o método descrito, devemos achar uma lista circular com oito vértices, correspondente à borda exterior. Uma vez achada esta lista, a outra lista da mesma componente conexa deve conter exatamente quatro vértices. Se estas condições forem preenchidas, podemos considerar que encontramos a componente do padrão. A probabilidade de se encontrar duas listas nestas condições e que não seja o padrão é, na prática, quase zero.

Encontrado os vértices do padrão, o próximo passo é identificar a qual vértice do modelo cada vértice encontrado corresponde. Novamente devemos começar pelos vértices da borda exterior. Os vértices encontrados formam

também uma lista circular, pois mantêm a ordem em relação um ao outro da lista de pixels original. Esta lista pode estar no sentido horário ou anti-horário. Consideremos que esteja no sentido horário, como está enumerado na figura 15. Queremos achar qual vértice corresponde ao vértice 0 na lista de vértices. Uma vez encontrado o vértice 0 ou qualquer outro vértice, basta seguir a ordem da lista para encontrar sucessivamente os próximos vértices.

Dado um vértice *m* na lista circular de vértices com $n = 8$, a distância entre este vértice e um outro vértice *p* continua sendo dado pelo mesmo vetor em (3.21). Porém podemos definir o seguinte produto vetorial entre vetores no $\Re^3$:

$$\overrightarrow{a_i} = (x_{i+1} - x_i, y_{i+1} - y_i, 0) \times (x_{i-1} - x_i, y_{i-1} - y_i, 0) \tag{3.23}$$

Para cada vértice nesta lista, podemos calcular o produto vetorial $\overrightarrow{a_m}$ correspondente. O resultado deste produto será um vetor com a componente *z* negativa, exceto para os vértices 4 e 5. Uma vez achados estes vértices nesta mesma ordem, podemos fazer a correspondência dos outros vértices.

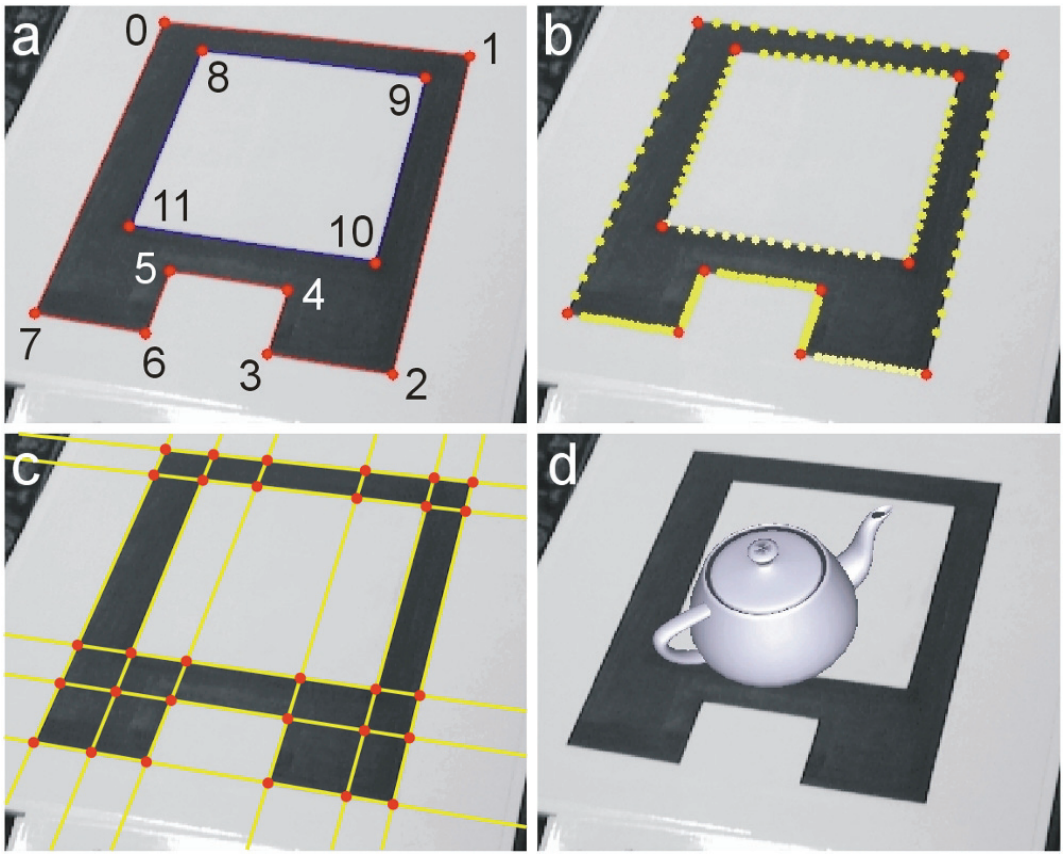

Figura 18 – Processo de obtenção dos vértices do padrão. (a) vértices identificados; (b) pontos das arestas; (c) retas encontradas e suas interseções; (d) resultado final.

Com os vértices encontrados da borda exterior, fica simples relacionar os vértices da borda interior. Basta achar o vértice 8 como sendo o mais próximo do vértice 0 da borda exterior e seguir a ordem na lista circular da borda interna. Com isso obtemos todos os pontos vértices já numerados, como mostra a figura 18a.

Porém apenas encontrar os pixels correspondentes a cada vértice no modelo não basta, pois a posição dos vértices estimada pelo procedimento acima está muito sujeita a erros e variações de uma imagem capturada para a próxima.

Para cada aresta entre um vértice e o seu subseqüente, podemos encontrar pixels pertencentes a esta aresta na lista de pixels de onde foram extraídos os vértices. Utilizando t pixels entre os dois vértices da aresta e igualmente espaçados um do outro, podemos calcular a melhor reta que passa por todos estes pontos, empregando o método dos mínimos quadrados. Se não houver t pixels, podemos utilizar todos os pixels entre os vértices. Então este método tenta minimizar $\sum_{i=1}^{t} \rho(r_i)$. A menor distância entre um pixel de índice i e a reta é dada por r_i. A função ρ é dada por:

$$\rho(r_i) = \frac{r_i^2}{2} \tag{3.24}$$

A figura 18b mostra os pixels das arestas utilizados, com $t = 14$ e a figura 18c mostra as retas encontradas para cada aresta. A partir das retas podemos calcular a interseção entre elas e reencontrar os valores dos vértices com maior precisão. Também teremos um número maior de pontos para o modelo, já que há mais interseções entre as retas do que vértices. Na última figura 18d temos o resultado da calibração, com um objeto virtual inserido na imagem capturada.

Este padrão de calibração confere grande flexibilidade no seu posicionamento em relação à câmera e outros objetos capturados na imagem não interferem no resultado. O único ponto contra é que ele deve permanecer integralmente visível na imagem capturada.

Szenberg [20] apresenta um estudo de métodos de calibração de câmera, onde ele conclui que o método de Tsai não apresenta bons resultados quando o número de pontos de calibração é baixo. Para resolver este problema, pode-se utilizar uma homografia na qual é definido um mapeamento do plano $z = 0$ para o plano da imagem, para estimar uma quantidade maior de pontos para serem

utilizados na calibração. Este método é utilizado na implementação da calibração de câmeras.

3.2. Luz Estruturada Codificada

Como foi visto no segundo capítulo, em nosso sistema estéreo ativo utilizamos luz estruturada codificada para fazer a correspondência entre pontos das imagens capturadas pelas duas câmeras. Na codificação temporal utilizada, projetamos uma seqüência de padrões com listras. A seqüência de n padrões produz 2^n listras codificadas e a resolução cresce exponencialmente com o número de padrões utilizados. Repetimos a mesma seqüência de padrões, porém com listras horizontais. Cada eixo precisa ser codificado separadamente em nosso sistema. Ao final teremos dividido a cena estática em $2^n \times 2^n$ regiões, cada uma com seu código horizontal e vertical.

Devemos distinguir entre as coordenadas das imagens capturadas de cada câmera, dadas por (x_e, y_e) para a primeira câmera e (x_d, y_d) para a segunda câmera, e as coordenadas da luz estruturada projetada, dadas por (u, v). Utilizamos a seqüência de imagens com listras horizontais e verticais para decodificar o código das coordenadas (u, v) para cada pixel das duas câmeras. Então para cada par de coordenadas (u, v) da luz estruturada, encontramos em cada câmera o grupo de pixels que tenham codificado o mesmo par de coordenadas, resolvendo o problema de correlação entre as imagens. Na prática, cada grupo possui mais de um pixel com o mesmo código.

Na codificação temporal binária o código traduzido pela seqüência de padrões é a própria numeração na base dois. Esta seqüência produz uma inconveniência à geração do código de cada linha. No objeto iluminado, existem regiões que jazem na fronteira entre uma listra branca e uma listra escura. Pixels nestas regiões devem pertencer a apenas uma das listras da fronteira. Um pixel não pode ser considerado de uma listra na imagem de um padrão projetado e posteriormente, na imagem do próximo padrão, pertencer a uma outra listra. Caso contrário acabaríamos com um código inválido para este pixel. Quando temos repetições de fronteiras, como no caso do código apresentado, fatalmente iremos ter este problema ou deveremos descartar os pixels próximos às fronteiras. Por

exemplo, um pixel deveria ter o código 011. Porém na imagem do primeiro padrão foi considerado pertencente à listra clara. Logo seu código ficou 111, indicando erroneamente que este pixel pertence a uma listra que nem é sua vizinha.

Uma maneira de resolver este problema é utilizar um dos códigos binários de Gray, conhecido como código binário refletido de Gray, ao invés da codificação puramente binária. Este código faz parte de um conjunto de códigos patenteados por Frank Gray, pesquisador da Bell Labs, em 1953. Porém estes códigos binários já eram utilizados desde o século XIX em diversas aplicações como o telégrafo, e sua origem pode ser estudada em [22]. A sua utilização foi proposta por [8] em substituição ao código binário. A figura 19 ilustra o código de Gray utilizado. Ele é chamado de código binário refletido, pois cada padrão é igual ao padrão anterior mais ele mesmo refletido e adicionado ao final. Na prática, o que interessa é que cada fronteira entre duas listras vizinhas só aparece uma vez, como é mostrado para a única fronteira do padrão. Além de evitar o problema de troca de código para todos os pixels nas regiões de fronteiras, esta codificação possui outra vantagem. Com exceção da primeira listra e da última listra do último padrão, todas as áreas claras ou escuras projetadas possuem pelo menos duas listras de largura. Quando o número de linhas cresce, acaba ocorrendo que as linhas ficam muito próximas uma das outras na imagem capturada, o que gera erro de amostragem. Utilizando o código de Gray, podemos diminuir este problema para um mesmo número de linhas em comparação com o código binário.

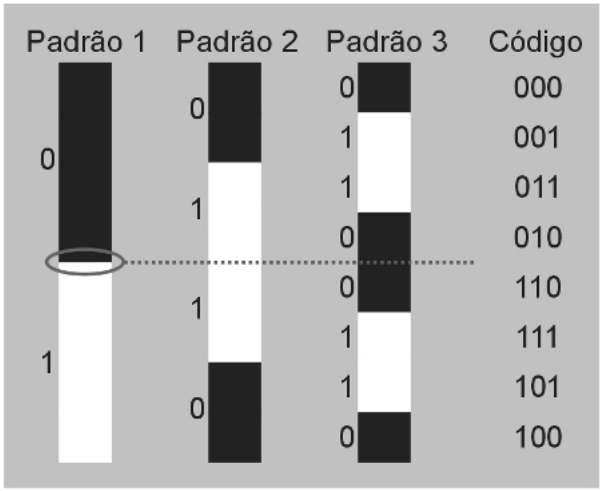

Figura 19 – Codificação temporal de Gray.

A conversão de números binários para o código de Gray correspondente é bastante simples. Dado um número binário $A_0\ A_1\ A_2 \ldots A_m$ contendo *m* dígitos onde A_0 é o dígito mais significativo, o código binário de Gray referente a este número é dado por $B_0\ B_1\ B_2 \ldots B_m$ onde:

$$B_0 = A_0$$

$$B_i = A_{i+1} \wedge A_i$$

A conversão do código de Gray para o número binário correspondente é dada por:

$$A_0 = B_0$$

$$A_i = ((\ldots (((B_0 \wedge B_1) \wedge B_2) \wedge B_3 \ldots) \wedge B_{i-1}) \wedge B_i$$

Devemos fazer algumas considerações a respeito da luz estruturada projetada. Como foi visto na introdução, esta luz é gerada por um projetor digital ligado a uma das saídas de vídeo do computador. Cada projetor possui uma resolução nativa dada pelo número de elementos que formam os pixels na imagem projetada. Devemos configurar a saída de vídeo com esta mesma resolução, para evitar uma re-amostragem da imagem pelos circuitos internos do projetor. Essa operação suaviza as bordas da imagem projetada, prejudicando assim as áreas de fronteiras entre as listras da luz estruturada.

Gostaríamos que cada listra possuísse a mesma largura na imagem projetada, ou seja, a largura de todas as listras deve possuir o mesmo número de pixel. Por exemplo, consideremos um projetor que possui uma resolução nativa de 800 pixels na horizontal. Se utilizarmos 128 listras verticais, então cada listra deve ter seis pixels de largura, que é igual ao maior inteiro menor que o resultado da divisão da resolução nativa pelo número de listras da luz estruturada. Na maioria dos casos, parte da imagem projetada não será utilizada.

Por último devemos ver a questão do processamento das listras claras e escuras nas imagens capturadas. O próximo capítulo mostra como é determinado o código binário horizontal e vertical de cada pixel a partir das imagens capturadas.

3.3. Captura e Processamento de Imagem

Antes da calibração de câmera e utilização do sistema estudado, é necessário tratar ruídos na imagem capturada. Tipicamente em câmeras digitais de vídeo encontramos ruído Gaussiano. Se considerarmos que o ruído introduzido na imagem é um sinal $n(x, y)$ adicionado ao valor real $C(x, y)$ do pixel na posição x e y da imagem, temos então que o valor de cada pixel na imagem é dado por:

$$E(x, y) = C(x, y) + n(x, y) \tag{3.25}$$

No caso do ruído Gaussiano, $n(x, y)$ é modelado por um processo estocástico Gaussiano de média zero. Podemos pensar que $n(x, y)$ é uma variável aleatória, distribuída de acordo com a função de distribuição Gaussiana de média zero, o qual é adicionado a cada pixel na imagem e cujos valores são independentes da posição e tempo. A função de distribuição Gaussiana de média igual a zero é dada por:

$$G(x) = \frac{1}{\sqrt{2\pi\sigma}} e^{-\frac{x^2}{2\sigma^2}} \tag{3.26}$$

O desvio padrão da distribuição é dado por σ e a figura abaixo ilustra um exemplo desta distribuição.

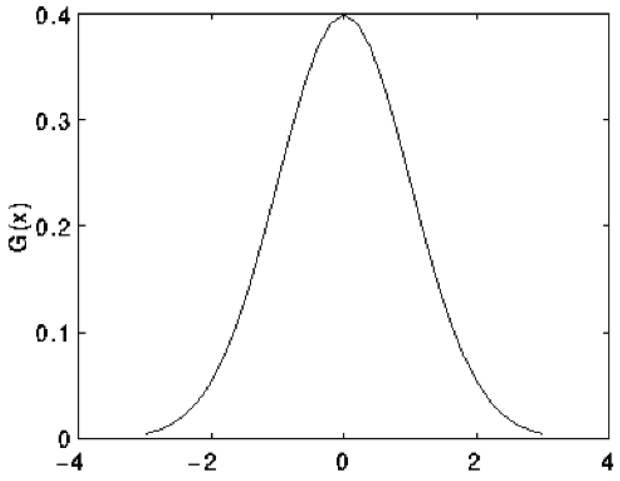

Figura 20 – Distribuição Gaussiana com $\sigma = 1$.

Uma das maneiras de tratar este ruído é aplicar o filtro de suavização Gaussiana à imagem capturada, como foi feito em [23]. Uma alternativa é capturar k diferentes imagens de uma mesma cena e retornar a imagem média de

todas elas, como mostra a equação (3.27). A cena deve ser estática e a iluminação não pode alterar. As imagens seriam idênticas se não houvesse ruído Gaussiano introduzido pelas câmeras. Então fazer uma média das imagens é igual a fazer a média dos sinais $n(x, y)$ do ruído e somar ao valor real $C(x, y)$ de cada pixel na imagem. Como $n(x, y)$ é distribuída de acordo com a Gaussiana de média zero, quanto mais imagens forem utilizadas na média, mais $n(x, y)$ se aproximará de zero. Logo a imagem terá menos ruído, ou seja, o ruído terá um menor valor em cada pixel, sem suavizar a imagem.

$$\overline{E(x, y)} = \frac{1}{k} \sum_{i=0}^{k-1} E_i(x, y) \tag{3.27}$$

Foi observado na prática que realmente quanto mais imagens utilizadas para a média, menor é o ruído Gaussiano. O valor padrão utilizado no sistema é $k = 10$.

Uma vez capturadas todas as imagens relativas aos padrões do código de Gray para cada câmera, o passo seguinte é identificar as listras claras e escuras dos padrões projetados sobre a cena em cada imagem. Mais precisamente, para cada pixel da imagem referente ao objeto capturado, devemos decidir se este pixel faz parte de uma listra clara ou de uma listra escura. Há também regiões do objeto capturado que não é iluminado pela luz estruturada, existindo assim mais uma opção. Regiões capturadas pelas câmeras, mas que não são iluminadas pelo projetor, são chamadas de regiões de sombra.

A figura 4 apresenta um exemplo de imagem capturada que deve ser tratada. Alguns pontos devem ser levados em consideração ao se resolver este problema. O valor de cada pixel depende da irradiação da imagem. Segundo o modelo Lambertiano de reflexão difusa, a radiação do objeto capturado, proveniente exclusivamente do projetor de luz, depende do valor de albedo do seu material. Também depende da inclinação da superfície em relação ao projetor. Porém devemos considerar que alguns materiais são especulares, como o material do vaso apresentado na figura 4, e refletem diretamente a luz projetada.

Uma maneira precisa de se resolver este problema foi utilizada em [3] e proposta inicialmente em [7]. Para cada padrão projetado, projeta-se também o seu inverso, que contém o mesmo número de linhas, porém as linhas que eram claras passam a ser escuras e vice-versa. A figura 21 mostra o par de imagens

capturados de um vaso iluminado por um padrão e o padrão inverso correspondente. Para determinar o caso de cada pixel utilizamos o seguinte filtro:

$$dst(x,y) = \begin{cases} preto, \quad se \quad dif > threshold \quad e \quad src(x,y) < \overline{src}(x,y) \\ branco, \quad se \quad dif > threshold \quad e \quad src(x,y) >= \overline{src}(x,y) \\ vermelho, \quad se \quad dif < threshold \quad e \quad mean < threshold2 \\ azul, \quad se \quad dif < threshold \quad e \quad mean >= threshold2 \end{cases} \tag{3.29}$$

$$dif = |src(x,y) - \overline{src}(x,y)|$$

$$mean = \frac{src(x,y) + \overline{src}(x,y)}{2}$$

A imagem capturada do padrão é dada por src, enquanto a imagem do padrão invertido é dada por $\overline{src}$. A imagem resultante é dada por *dst*, cujos pixels possui um dos quatro valores possíveis. Cada valor é representado por uma cor, para melhor visualização do resultado na figura 22. O primeiro passo do filtro é determinar se o módulo da diferença do valor do pixel nas duas imagens é maior que um valor de corte, dado por *threshold*. No caso positivo, consideramos que a diferença nas imagens entre o padrão projetado e seu inverso correspondente projetado é suficientemente grande neste pixel para determinar se este pertence a uma listra clara ou escura. Caso o seu valor na primeira imagem seja maior que na segunda, correspondente à imagem capturada do padrão invertido, este pixel pertence a uma listra clara. No sistema implementado, o valor para *threshold* comumente utilizado é 30 em uma escala de tons de cinza de 8 bits. O valor branco é atribuído a pixels pertencentes a listras claras, enquanto preto é atribuído a pixels pertencentes a listras escuras.

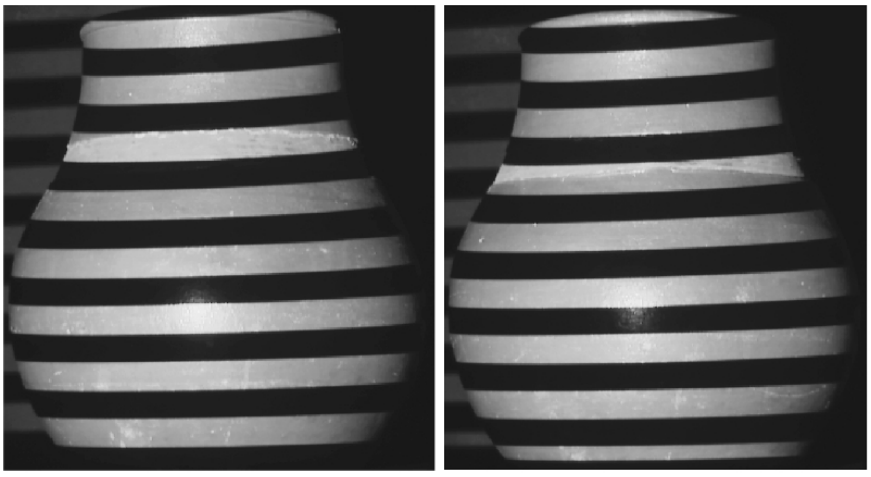

Figura 21 – Padrão e seu inverso projetados sobre objeto.

Nos outros dois casos, onde a diferença do pixel nas duas imagens não é suficientemente grande, o pixel não é atribuído a nenhuma listra. Podemos estabelecer um segundo valor de corte, dado por *threshold2*, cujo valor é tipicamente 128, correspondente à metade da escala de tons de cinza. Caso a média dos valores de um pixel seja menor que *threshold2*, atribui-se o valor vermelho. Usualmente pixels com este valor representam regiões do objeto não iluminadas pela luz estruturada, a parede de fundo ou ainda regiões de transição entre uma faixa clara e outra escura. Caso a média seja maior que *threshold2*, atribui-se o valor azul. Comumente pixels com este valor representam regiões de transição entre duas faixas de tonalidades diferentes ou regiões do objeto que refletem a luz estruturada do projetor.

Figura 22 – Resultado do filtro para determinar listras projetadas.

O ponto positivo de se utilizar esta técnica é que tornamos a detecção de listras mais robustas, independente do material e da inclinação de superfícies do objeto capturado em relação ao eixo de projeção. Esta técnica também se mostrou tolerante a superfícies moderadamente especulares. O ponto negativo é que dobramos o número de imagens capturadas.

Infelizmente há casos em que utilizar padrões e seus inversos não é suficiente para distinguir as listras em superfícies com grande variação de albedo. Por exemplo, podemos querer modelar o prato mostrado na figura 23a e 23b. Porém os desenhos artísticos pintados sobre o prato dificultam determinar as listras do padrão projetado nestas imagens. O resultado está ilustrado na figura 23c. Em outro caso, uma superfície pode estar bastante inclinada em relação ao

eixo de projeção, também dificultando a distinção das listras. Na prática, para resolver estes problemas é necessário capturar as imagens utilizando dois valores de exposição diferentes. Por exemplo, se capturarmos a imagem de um padrão utilizando uma exposição de 0,1*s*, então podemos capturar outra imagem correspondente com uma exposição de 0,5*s*. Aumentando a exposição, tornamos as áreas escuras da imagem mais claras e saturamos as áreas claras. Para cada pixel, aplicamos (3.29) ao par de imagens da exposição que tenha a maior diferença absoluta entre eles. Assim podemos distinguir as listras em diferentes tipos de superfícies. As figuras 23d e 23e ilustram o par de imagens capturadas com uma maior exposição. Utilizando ambos os pares de imagens obtemos um melhor resultado, apresentado na figura 23f.

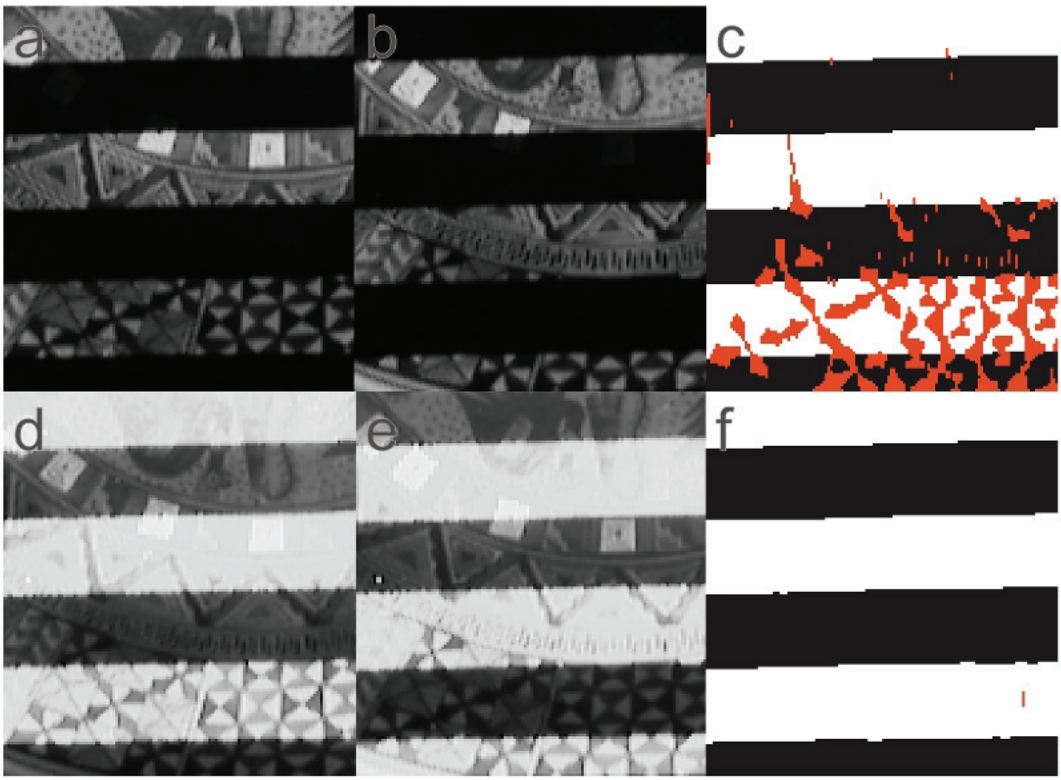

Figura 23 – Captura com diferentes valores de exposição: (a) imagem do padrão; (b) imagem do padrão inverso; (c) resultado para um par de imagens; (d) imagem com alta exposição do padrão; (e) imagem com alta exposição do padrão inverso; (f) resultado para dois pares de imagens.

Teoricamente poderíamos utilizar mais de dois valores de exposição para cada objeto modelado. Porém dois valores se mostraram suficientes, bastando apenas um na maioria dos objetos. Novamente, o ponto negativo deste recurso é que para cada padrão projetado devemos capturar mais um par de imagens para

cada valor de exposição utilizado, aumentando o tempo total de captura. Uma abordagem semelhante foi utilizada em [3].

Dos resultados obtidos podemos notar que entre duas listras há sempre uma fina listra de pixels atribuídos a nenhuma delas. Estes pixels se situam exatamente na fronteira entre duas listras. Como foi visto, cada fronteira aparece apenas uma vez na codificação de Gray utilizada. Isto permite tratar estes pixels de maneira a determinar a qual das duas listras de uma fronteira um pixel de valor azul ou vermelho pertence, sem gerar inconsistência na sua codificação. Podemos definir que um pixel pertence a uma região de fronteira entre duas listras se este possuir valor vermelho ou azul e se seus vizinhos pertencerem a uma determinada listra. Em imagens com listras horizontais verificamos para cada pixel, cujo valor binário não está definido, os seus vizinhos verticais acima e abaixo, enquanto que em imagens com listras verticais verificamos os seus vizinhos horizontais na esquerda e direita. Não basta verificar apenas os vizinhos diretamente adjacentes ao pixel verificado. Devemos verificar n pixels em cada direção. Por exemplo, para um pixel em uma imagem com listras verticais, devemos verificar os seus n vizinhos mais próximos à esquerda e seus n vizinhos mais próximos à direita. Na prática utilizou-se $n = 3$. Se todos estes $2n$ vizinhos pertencerem a alguma listra, então podemos forçar este pixel também a pertencer a uma destas listras. Caso os seus vizinhos à esquerda pertençam a uma listra distinta da dos seus vizinhos à direita, o pixel analisado pertence a uma região de fronteira indefinida. Podemos aplicar (3.29) novamente, porém considerando que $threshold = -1$, de modo a forçar o pixel a escolher uma da duas listras adjacentes. Caso todos os seus vizinhos pertençam à mesma listra, o pixel não está em uma fronteira, mas dentro de uma listra e deve possuir o mesmo valor que seus vizinhos.

Figura 24 – Detecção e refinamento de fronteiras.

Em ambos os casos não há perigo de um pixel acabar pertencendo a uma listra da qual não é vizinho. Podemos estender a idéia para mais vizinhos de um pixel, alargando a área considerada uma fronteira. A figura 24 mostra o resultado desta operação considerando fronteiras com dois pixels de largura.

No final de todas as operações mencionadas até agora nesta seção, obteremos um conjunto de 2*n* imagens, codificadas por uma luz estruturada de *n* padrões com listras verticais e *n* padrões com listras horizontais. Um pixel terá um valor codificado se este possuir valor branco ou preto em todas as 2*n* imagens. Caso o seu valor não esteja determinado em uma ou mais imagens, não é possível determinar univocamente a qual listra este pixel pertence, tanto verticalmente quanto horizontalmente. Para cada pixel na imagem determina-se, caso seja possível, os dois valores codificados para posteriormente convertê-los do código de Gray utilizado a números binários correspondentes. Determinamos assim a quais listras horizontal e vertical cada pixel pertence. Repetimos o processo para a outra câmera.

3.4. Triangulação e Modelo Final

Feita a correspondência entre pixels das duas câmeras calibradas, gostaríamos de obter uma nuvem de pontos do objeto modelado. Para cada par de coordenadas (u,v) da luz estruturada codificada, encontramos em cada imagem os pixels que codificam este par. Usualmente teremos um grupo de pixels conexos em cada imagem. A figura 25 mostra os grupos encontrados, cada um com uma cor, para uma luz estruturada de 32 listras verticais e 32 listras horizontais. Note que a luz estruturada iluminou também a parede no fundo. De um outro ponto de vista podemos dizer que estamos dividindo o objeto em diversos pedaços menores.

Para a triangulação necessitamos de apenas um ponto que melhor representa cada grupo nas duas imagens. Dentre as tentativas de se obter este ponto, a que melhor apresentou resultados foi utilizar o centróide de cada grupo. Calcula-se a coordenada média na imagem utilizando-se todos as coordenadas dos pontos de cada grupo.

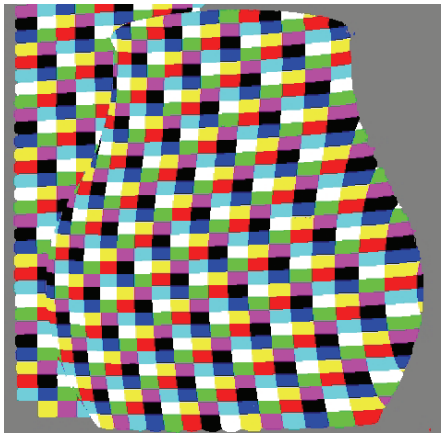

Figura 25 – Grupos de pixels com mesmas coordenadas codificadas.

Se o número de listras verticais e horizontais é suficientemente grande, podemos considerar que cada grupo de pixels no objeto dividido, cuja forma na imagem se aproxima a de um retângulo, representa um polígono quadrilátero no espaço. A projeção do seu ponto central, dada pela interseção das diagonais do retângulo na imagem, seria um melhor ponto representativo. Esta segunda abordagem também foi utilizada, porém apresentou resultados inferiores aos obtidos pela primeira.

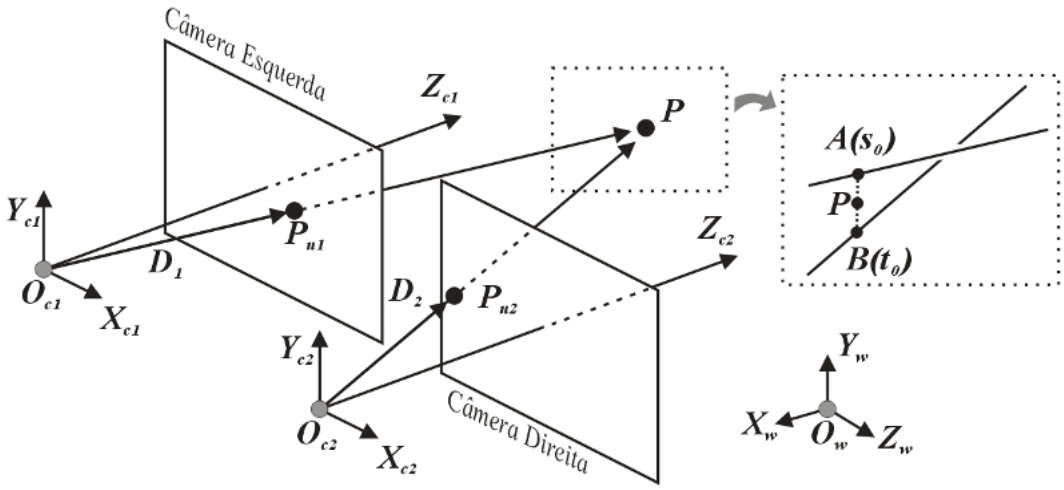

Figura 26 – Triangulação e interseção de duas retas no espaço.

Para cada câmera, calcula-se a reta que sai do seu centro de projeção O_c e intercepta o plano da imagem no ponto médio P_u calculado. A seção 3.2 descreve como calcular P_u a partir do ponto médio na imagem capturada. A reta continua e intercepta a reta da outra câmera no ponto P. Teoricamente a probabilidade de

duas retas se interceptarem no espaço é zero, mesmo para retas arbitrariamente próximas. Então P é dado pelo ponto médio do segmento de reta que liga cada ponto em cada reta mais próximo da outra, como mostra em detalhe a figura 26.

Podemos escrever as equações paramétricas das retas da câmera esquerda e direita como sendo, respectivamente:

$$A(s) = O_{c1} + s(P_{u1} - O_{c1}) = O_{c1} + sD_1 \tag{3.30}$$

$$B(t) = O_{c2} + t(P_{u2} - O_{c2}) = O_{c2} + tD_2 \tag{3.31}$$

Os pontos utilizados devem estar de acordo em relação ao sistema de coordenadas, convertendo todos para o sistema de coordenadas do mundo dado por (X_w, Y_w, Z_w). Podemos definir o seguinte vetor que dá a distância entre um ponto na primeira reta a um segundo ponto na outra reta:

$$W(s,t) = A(s) - B(t) \tag{3.32}$$

Substituindo (3.30) e (3.31) em (3.32) e declarando $O_w = O_{c2} - O_{c1}$, temos:

$$W(s,t) = O_w + sD_1 - tD_2 \tag{3.33}$$

Existe apenas um par de pontos, dados por $A(s_0)$ e $B(t_0)$, onde $W(s,t)$ possui tamanho mínimo. Queremos achar o valor de s_0 e t_0. Se as duas retas não são paralelas, o que sempre é verdadeiro no sistema quando não há erros de calibração ou de correspondência entre pontos, é fácil perceber que o seguimento de reta que une o par de pontos $A(s_0)$ e $B(t_0)$ é perpendicular às direções das retas, caso contrário não seriam os pontos mais próximos. As seguintes equações, envolvendo produto escalar, são verdadeiras:

$$D_1 \cdot W(s_0, t_0) = 0 \tag{3.34}$$

$$D_2 \cdot W(s_0, t_0) = 0 \tag{3.35}$$

Substituindo (3.33) em (3.34) e (3.35) obtemos:

$$(D_1 \cdot O_w) + s_0(D_1 \cdot D_1) - t_0(D_1 \cdot D_2) = 0$$

$$(D_2 \cdot O_w) + s_0(D_2 \cdot D_1) - t_0(D_2 \cdot D_2) = 0$$

Fazendo $a = D_1 \cdot D_1$, $b = D_1 \cdot D_2$, $c = D_2 \cdot D_2$, $d = D_1 \cdot O_w$ e $e = D_2 \cdot O_w$, encontramos:

$$s_0 = \frac{be - cd}{ac - b^2} \tag{3.36}$$

$$t_0 = \frac{ae - bd}{ac - b^2} \tag{3.37}$$

Os denominadores de (3.36) e (3.37) são diferentes de zero quando as retas não são paralelas. A posição de P é dado por:

$$P = B(t_0) + \frac{W(s_0, t_0)}{2} \tag{3.38}$$

Repetindo o cálculo para todas as coordenadas iremos obter uma nuvem de pontos distribuídas no espaço correspondente ao modelo. Na prática sempre serão incluídos pontos da parede de fundo ao modelo capturado, principalmente quanto utilizamos valores altos de exposição para as câmeras. Podemos resolver este problema criando um plano de corte, que descarta todos os pontos encontrados situados além deste, e o posicionamos entre o objeto capturado e a parede de modo a descartar estes pontos indesejados. A figura 27 mostra a nuvem de pontos modelada a partir do vaso utilizado como modelo em duas vistas distintas. Os pontos de fundo foram retirados com um plano de corte.

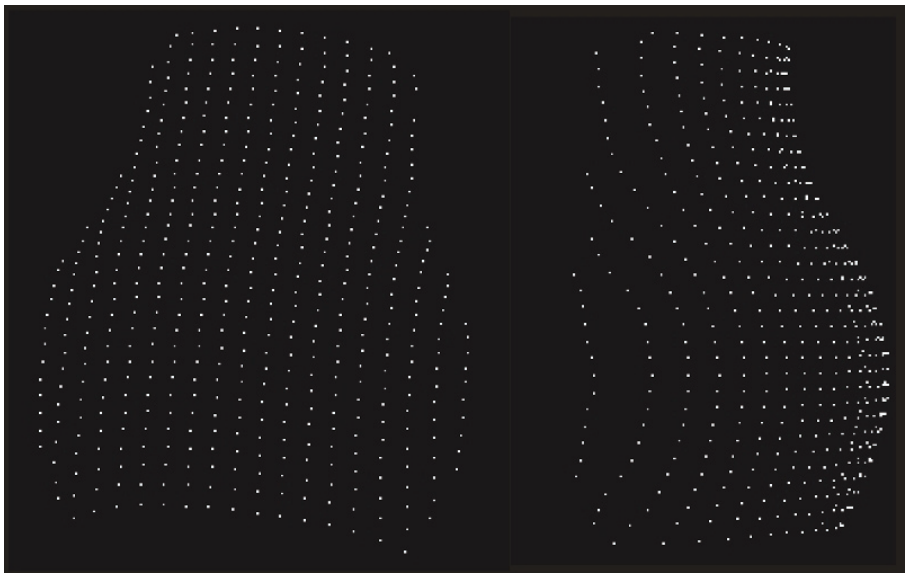

Figura 27 – Nuvem de pontos do objeto modelado.

Utilizando estes pontos como vértices, gostaríamos de obter um modelo poligonal do objeto capturado. Entre outras possibilidades, podemos criar polígonos com vértices próximos um do outro. Porém, ao invés de considerar vértices vizinhos geométricos, escolhemos vértices que possuem coordenadas codificadas (u, v) vizinhas. Por exemplo, podemos criar um polígono com os vértices de coordenadas $(25,37)$, $(26,37)$ e $(25,38)$. A normal em cada vértice é

calculada utilizando-se os oito vértices vizinhos. Calculam-se as normais dos oito polígonos triangulares que possuem este vértice. Estas normais são somadas e o vetor resultante normalizado é utilizado como a normal deste vértice. O resultado desta operação aplicada aos pontos da figura anterior está apresentado na figura 28a.

Para a textura do modelo, captura-se uma imagem do objeto com cada uma das câmeras separadamente da seqüência de imagens capturadas dos padrões projetados. O objeto deve estar iluminado por luz ambiente, de maneira que suas superfícies visíveis estejam uniformemente iluminadas. Com isso obtemos texturas de melhor qualidade do que as obtidas apenas iluminando o objeto capturado com a luz do projetor.

A coordenada de textura para cada vértice é dada pela sua posição média na imagem da câmera, já calculada anteriormente na etapa de triangulação. Então teremos para cada câmera uma textura e mapeamento correspondente. Ambas as texturas são usadas simultaneamente utilizando multi-textura, suportada pela grande maioria das placas gráficas atuais. A figura 28b apresenta o modelo com textura aplicada.

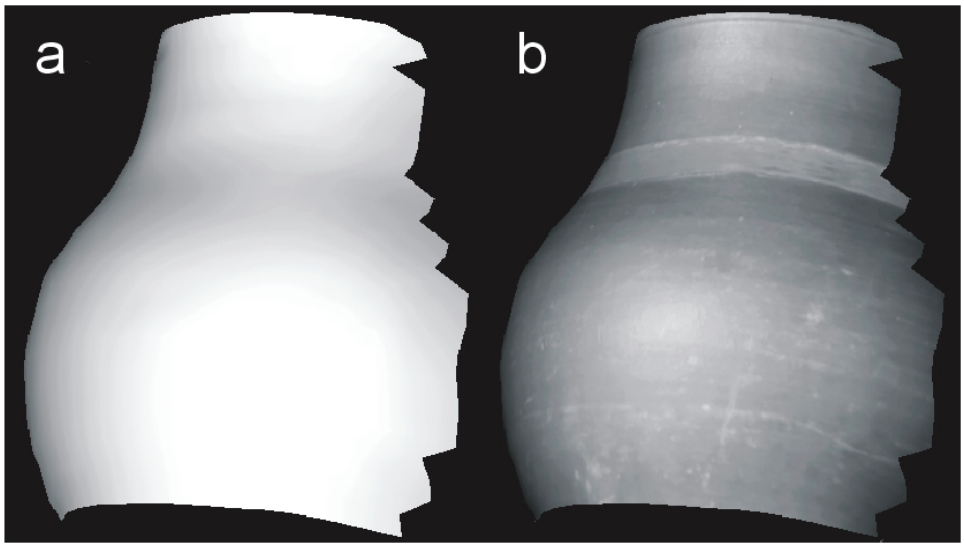

Figura 28 – Modelo geométrico poligonal criado: (a) sem textura; (c) com textura.