DOI: 10.48011/asba.v2i1.1605

Pesquisa de Posição Ocular por Correspondência de Modelos para Renderização Baseada em Fóvea com FPGA

Gabriel Ayres de Oliveira * **Estêvão Coelho Teixeira** **

Faculdade de Engenharia, Universidade Federal de Juiz de Fora, MG

* *(e-mail: gabriel.oliveira@engenharia.ufjf.br)*
** *(e-mail: estevao.teixeira@ufjf.edu.br)*

Abstract: The continuous improvement of virtual reality headsets, with resolution and frame rate growth, has been making frame rendering very intensive for current available hardware. To decrease this overhead, new techniques such as foveated rendering are used, where frames are rendered with different fidelity levels mimicking the physiology of the human eye. This type of rendering, on the other hand, needs eye tracking. The work presented here focuses on tracking the user's gaze using a template matching technique applied to a dedicated hardware implementation on an FPGA.

Resumo: O contínuo melhoramento dos *headsets* de realidade virtual, tem levado a um crescente aumento de resolução e taxa de atualização, tem tornado a geração de quadros cada vez mais custosa para o hardware disponível atualmente. Uma das opções para mitigar esse custo computacional utiliza técnicas como a renderização foveada, em que partes da imagem têm maior ou menor definição, imitando a fisiologia do olho humano. Visto que visão humana não é fixa em um ponto, faz-se então necessário buscar a posição ocular. Este trabalho tem por objetivo a proposta de uma técnica para encontrar essa posição usando um hardware dedicado implementado em FPGA.

Keywords: eye tracking, foveated rendering, image processing, FPGA, template matching

Palavras-chaves: pesquisa posição, renderização foveada, processamento de imagem, FPGA, correspondência de modelos

1. INTRODUÇÃO

Os *headsets* de realidade virtual (Sutherland, 1968), que por muito tempo tiveram seu desenvolvimento estagnado, vêm, contudo, apresentando um renovado interesse atualmente, que proporcionou grandes avanços nestes últimos anos. Em função disso, suas áreas correlatas têm sido alvo de constante melhoramento. Dentre essas estão, por exemplo, aparatos óticos, áudio, sensores de movimento e posição, e *displays* (Coburn et al., 2017). Com o objetivo de imergir o usuário em outra realidade, com estes dispositivos, busca-se tornar as imagens geradas fidedignas ao mundo real. Tal busca tem levado a um aumento de resolução, assim como um aumento da taxa de atualização dos painéis que os equipam. Usando-se técnicas de renderização convencionais, o esforço computacional para gerar esses quadros tem crescido consideravelmente. De forma a diminuí-lo, maneiras mais eficientes, como a renderização de fóvea vêm sendo utilizadas. Esta, imitando a fisiologia do olho humano, gera quadros com locais de maior ou menor fidelidade, efetivamente diminuindo o custo computacional dos mesmos (Guenter et al., 2012).

O olho humano não é estático. Assim, se faz necessário encontrar a sua posição para que as áreas de diferentes fidelidades do quadro sejam geradas de acordo. O trabalho aqui apresentado tem por objetivo implementar um *eye-tracker* para esse tipo de processo usando a técnica de correspondência de modelos. Esse processo é então aplicado em hardware dedicado descrito por Verilog em uma FPGA (*Field Programmable Gate Array*, Arranjo de Portas Programável em Campo), de maneira a desacoplar o esforço computacional do hardware principal.

A pesquisa feita difere de outros trabalhos disponíveis na literatura na maneira como a pesquisa de posição ocular é feita. Em Yan et al. (2009) a pupila é primeiramente segmentada usando uma imagem binária e em seguida um *fitting* é feito, aproximando-a de uma elipse, o que difere deste trabalho, onde o tratamento é feito em escala de cinza. A simplicidade desta implementação também difere de Li and Wee (2009), onde algumas transformações matemáticas são aplicadas à imagem antes da mesma ser analisada. Em ambos os casos, o hardware extra intrínseco às transformações e *fittings* torna ambas implementações custosas do ponto de vista computacional.

O restante do trabalho apresenta uma breve conceituação feita na Seção 2, explicando a renderização de fóvea e a correspondência de modelos. Na Seção 3, é descrita a implementação, onde a Subseção 3.1 explicita o hardware escolhido e suas conexões, a escolha de parâmetros dado as limitações do hardware escolhido na Subseção 3.2 e na Subseção 3.3 uma descrição dos módulos que compõem o

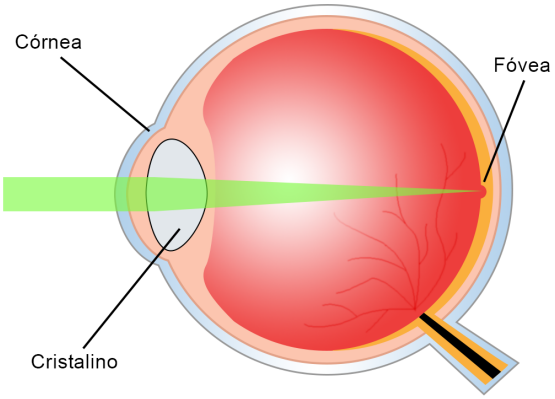

Figura 1. Seção transversal do olho humano, explicitando a posição da fóvea.

sistema, assim como seu funcionamento por meio de um fluxograma. A Seção 4 expõe os resultados encontrados ao fim do processo e também compara com outros trabalhos na literatura. Por fim, a conclusão é apresentada na Seção 5.

2. CONCEITUAÇÃO

2.1 Renderização Foveada

O campo de visão humano, apesar de largo, apresenta algumas particularidades em relação a suas extremidades. Com um alcance de 135 graus vertical e 160 graus horizontal, somente um cone de 5 graus partindo do centro da visão produz imagens de alta fidelidade (Kolb, 2011). A Figura 1 mostra um pequeno sulco na parte posterior dos olhos chamada fóvea. Nessa área estão localizados os cones, receptores dos três comprimentos de onda que formam as imagens que vemos. Uma característica da fóvea é a diminuição drástica na contagem de cones à medida que se afasta do seu centro. Para a visão humana, isso significa que as imagens projetadas nessa região têm definição mais alta em seu centro e mais baixa em sua periferia.

A renderização foveada utiliza-se dessa particularidade junto com um *eye-tracker* para gerar quadros com maior resolução perto na área da visão onde distingue-se mais detalhes, e com menos fidelidade nas áreas periféricas. Visto que imagens de menor resolução tem *pixels* visíveis quando expandidas, de modo a terem o mesmo tamanho da imagem original, processos de desfoque simples também são aplicados de forma a exibirem uma transição mais gradual entre as partes de alta e baixa resolução. Com esse tipo de técnica é possível obter cinco a seis vezes mais quadros por segundo do que em comparação a uma renderização com fidelidade constante em todo o quadro (Guenter et al., 2012). A Figura 2 mostra uma imagem com os diferentes níveis de fidelidade sobrepostos.

2.2 Correlação de imagens

A técnica usada nesse trabalho baseia-se na correspondência de modelos (Brunelli, 2009). Esta consiste em procurar dentro de uma imagem alguma posição que se assemelhe a um modelo previamente escolhido. Nesse processo todas as

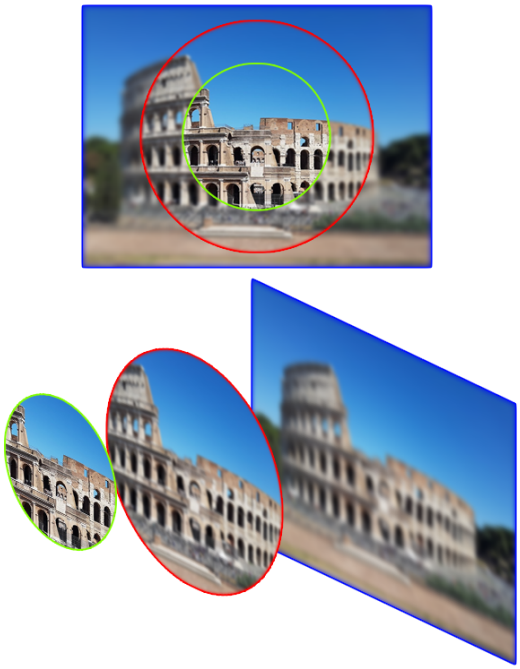

Figura 2. Quadro renderizado com diferentes fidelidades

posições de imagem são sobrepostas com o modelo separadamente, e uma nota é então dada para cada uma delas. A nota é calculada analisando dois a dois os *pixels* do modelo e os *pixels* correspondentes da imagem original sobrepostos pelo modelo (Latecki, 2005). A Figura 3 demonstra o deslocamento do modelo sobre a imagem. A escolha dessa técnica se dá pela sua simplicidade de implementação em hardware e bons resultados.

$$C_{(x,y)} = \sum_{x=0}^{x_{max}} \sum_{y=0}^{y_{max}} (1023 - |P_{imagem}(x,y) - P_{modelo}(x,y)|) \quad (1)$$

A equação 1 mostra o cálculo de correlação realizado. Aqui, $C_{(x,y)}$ é a nota da correlação entre a imagem pesquisada e o modelo. $P_{imagem}(x,y)$ e $P_{modelo}(x,y)$ são os dois *pixels* a serem comparados, da imagem e do modelo, respectivamente. O módulo da diferença entre os *pixels* tocados pelo modelo é feita dois a dois e diminuído de um valor arbitrário máximo que o *pixel* pode assumir. No

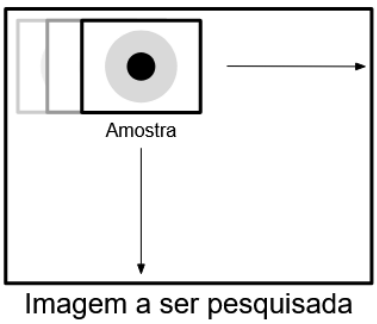

Figura 3. O modelo é varrido por todas as posições da imagem e comparado.

caso da implementação específica para a câmera escolhida, o maior valor que um *pixel* pode assumir é 1023. Este então é o valor arbitrário escolhido.

Para dois *pixels* iguais, sua comparação resulta numa nota de 1023. No caso de dois *pixels* completamente diferentes a nota seria zero. Um valor $C_{(x,y)}$ igual a zero significa que todos os *pixels* do modelo e da imagem original são diferentes. O maior valor que a correlação pode alcançar, porém depende da resolução do modelo, descrito pela equação 2.

$$C_{(x,y)} = 1023 \times x_{max} \times y_{max} \tag{2}$$

Aqui x_{max} e y_{max} são os valores de resolução vertical e horizontal do mesmo. No caso do modelo escolhido para esse trabalho, que tem como tamanho 256x256 *pixels*, o maior resultado possível seria então de 67.043.328.

3. IMPLEMENTAÇÃO

3.1 Hardware

A implementação proposta nesse trabalho foi feita em uma placa de desenvolvimento Terasic DE2-115. Essa placa está munida de uma FPGA Intel Cyclone IV (Anteriormente Altera) e também conta com alguns periféricos importantes usados aqui, como a SRAM (*Static Random-Access Memory*, Memória de Acesso Aleatório Estática) e a SDRAM (*Synchronous Dynamic Random-Access Memory*, Memória de Acesso Aleatório Dinâmica Síncrona). Essa placa também contém um conversor digital-analógico usado exclusivamente para a saída VGA (Terasic, 2010). A câmera usada, TRDB-D5M, também da Terasic, apresenta um sensor CMOS de 5 mega*pixels*. Essa câmera produz imagens de até 1024x768 *pixels* a 30 quadros por segundo (Terasic, 2017). As imagens são entregues à FPGA por um meio de barramento de dados de 10 bits, sincronizado por sinais de sincronia vertical (Vsync), horizontal (Hsync) e um clock específico (PXLclk). A FPGA, por sua vez, controla parâmetros da câmera como sua exposição e resolução através de um barramento I2C. Essas conexões estão explicitadas na Figura 4.

De maneira a aumentar o brilho da imagem, um anel de LEDs infravermelho foi colocado em volta da câmera. Dessa maneira a mesma pode ficar próxima ao olho do usuário sem que o mesmo sinta o desconforto de ser ofuscado. Essa montagem pode ser observada na Figura 5.

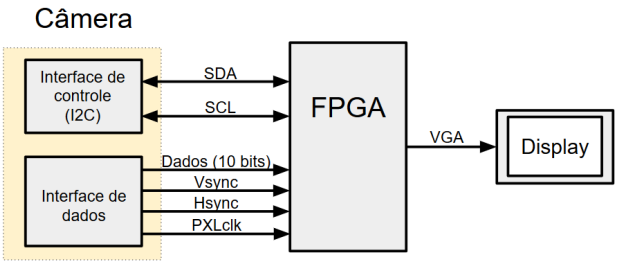

Figura 4. Conexões de dados e de controle entre a câmera e a FPGA.

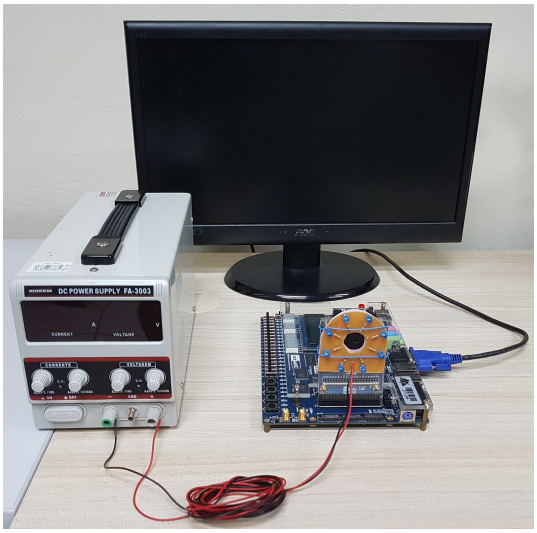

Figura 5. A montagem do sistema, com o anel de LEDs circundando a câmera.

3.2 Escolha de parâmetros

A exposição das escolhas de parâmetros aqui se faz necessária devido a alguns destes serem escolhas deliberadas e outros limitações impostas pela placa de desenvolvimento.

A correspondência de modelos é feita geralmente em escala de cinza ou monocromática como recomenda a literatura (Duchowski, 2009). Preferiu-se aqui a escala de cinza, de modo a preservar os detalhes da imagem. Outro motivo importante se dá pela iluminação infravermelha usada, não existindo então a necessidade de abranger todo o espectro. Limitando-se então a somente uma cor, diminui-se também o espaço usado pelo quadro analisado salvo na memória.

Visto que nesse trabalho a SDRAM já está sendo usada como *buffer* de vídeo para a saída VGA e salvar um quadro como unidades lógicas da FPGA consumiria muito espaço, o quadro a ser analisado é salvo na SRAM.

Cada um dos *pixels* é salvo em uma posição de memória. Com 10 bits para cada *pixel*, 6 bits são desperdiçados visto que o tamanho da palavra para essa memória é de 16 bits. O espaço usado está explicitado por S_{quadro} em porcentagem do total disponível na equação 3. A SRAM presente na placa possui 2MB de espaço distribuído como um milhão de palavras de 16 bits. A implementação usou uma imagem a ser pesquisada com resolução de 640x480 *pixels* e 16 bits por *pixel*. Em sua totalidade, esta ocupa aproximadamente 30% do espaço disponível. Apesar de parecer um desperdício de memória, essa maneira de armazenar simplifica o acesso, garantindo também que somente um acesso à memória precise ser feito para cada *pixel*.

$$S_{quadro} = \frac{640 \times 480 \times 16}{16 \times 10^6} = 0,30 = 30\% \tag{3}$$

A implementação atual possui um *clock* de 50 MHz. Visto que o processo de correlação de imagens requer muitos acessos, este é um dos fatores limitantes para a velocidade do sistema.

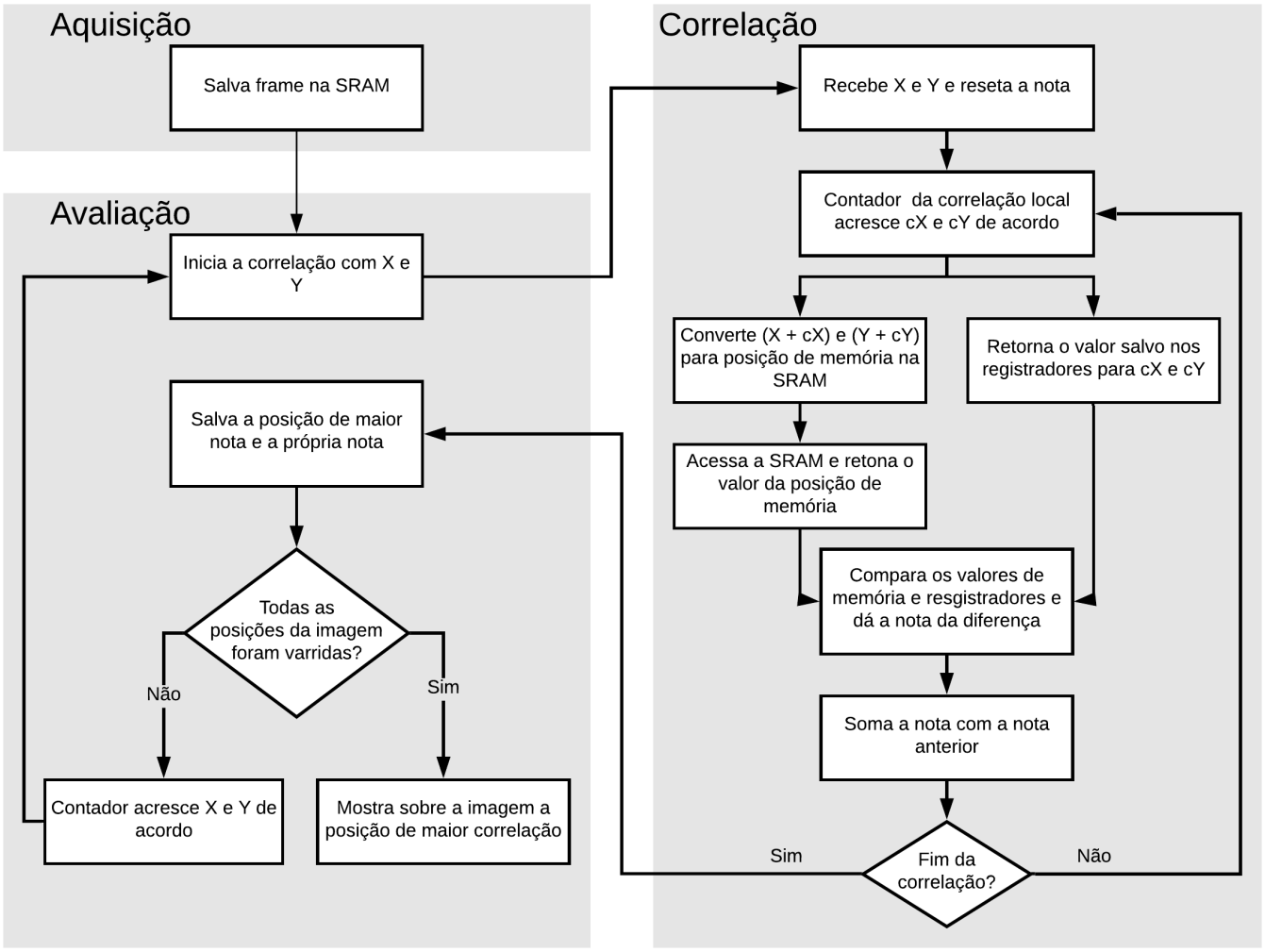

Figura 6. Diagrama de fluxo detalhando o funcionamento do sistema.

3.3 Módulos e Funcionamento

Todos os módulos que controlam esse processo foram escritos em Verilog. O diagrama mostrado na Figura 6 detalha o funcionamento do sistema, onde (X, Y) é a posição onde a correlação está sendo feita, sendo atualizado conforme o modelo varre a imagem original, e cX, cY a posição momentânea do processo de correlação por si, calculando a diferença entre o *pixel* do modelo e o *pixel* equivalente da imagem.

Captura e armazenagem da imagem De maneira a ordenar o processo, esse módulo está encarregado de registrar o quadro exibido pela câmera. Caso o sinal de controle esteja em nível alto, indicando que o processo pode ser iniciado, este módulo interrompe a captura da câmera após um tempo especificado previamente. A câmera, por sua vez alimenta continuamente o *buffer* da SDRAM com o último quadro capturado. Dessa maneira pode-se sobrepor o resultado da correlação do quadro sobre o mesmo no fim do processo.

Nessa implementação, o quadro capturado é proveniente da saída VGA. Sendo assim, deve-se ignorar a janela de sincronia deste sinal (*Sync, Back Porch e Front Porch*) visto que não representam informações acerca à imagem capturada. O quadro tem seus *pixels* armazenados na posição de memória de acordo com a equação 4.

$$P_{memoria} = iX - iX_{start} + 640 \times (iY - iY_{start}) \quad (4)$$

Nesta, iX e iY são os índices dos *pixels* sendo exibidos na saída VGA e são salvos, cada um, na posição de memória $P_{memoria}$. A contagem de *pixels* da sincronia horizontal e vertical estão representadas por iX_{start} e iY_{start} respectivamente, e são a soma dos os valores de *Sync*, *Back Porch* e *Front Porch* para a janela horizontal e vertical.

Armazenagem do modelo Um dos módulos mais importantes dessa implementação está no armazenador do modelo. Visto que a SDRAM já está sendo usada para o *buffer* da saída VGA, só resta usar a SRAM como local de armazenamento para o modelo. Dado que o processo de correlação de imagem realiza inúmeros acessos à SRAM para ler a imagem original, guardar também o modelo nesta memória faria com que para cada comparação *pixel*

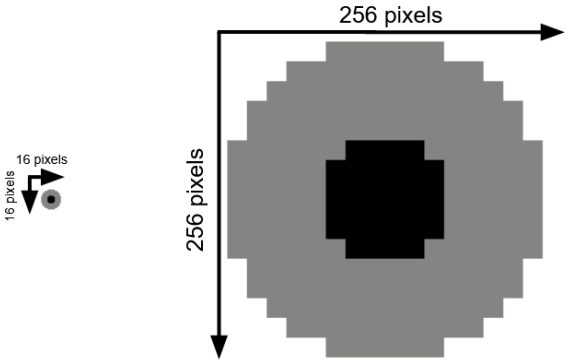

Figura 7. Comparação do modelo original salvo no módulo e sua versão ampliada.

a *pixel*, dois acessos seriam necessários, duplicando efetivamente o tempo total gasto pela implementação. A solução vem então de criar um módulo à parte onde essa imagem é armazenada. Caso fosse guardada em seu tamanho original de 256x256 *pixels*, ocuparia muito espaço da FPGA em termos de elementos lógicos. A solução é então armazenar uma imagem menor, tendo como escolha uma resolução de 16x16 *pixels* e então ampliá-la para a resolução desejada. A Figura 7 mostra as duas versões do modelo, armazenada e ampliada respectivamente.

Controle de Correlação O módulo de controle da correlação tem como objetivo sequenciar as posições da imagem original onde a correlação deve ser executada, iniciando exatamente após o fim da captura do quadro. Estas posições são passadas para o módulo de correlação de maneira que todas as posições sejam varridas. Esse módulo também faz o registro, no fim da correlação, das coordenadas e seu resultado caso este seja o maior calculado até o momento.

Correlação A correlação de imagens se dá pela comparação dois a dois dos *pixels* da imagem e do modelo, avaliando com uma nota a semelhança entre esses *pixels*. O módulo de correlação recebe do Controle de Correlação a posição onde o processo deverá ser realizado. Acessando a SRAM e o módulo do modelo, a correlação é executada para todos os *pixels* do modelo e da imagem original sobrepostos pelo modelo, retornando para o módulo de Controle a nota calculada. A avaliação é feita usando a equação (1).

Conversão em escala de cinza O módulo de conversão ajusta o quadro para escala de cinza antes de ser salvo na SRAM como explicitado anteriormente. Essa conversão é feita *pixel* a *pixel* como explicitada na equação 5,

$$P_{cinza} = P_r \times 0,30 + P_g \times 0,59 + P_b \times 0,11 \qquad (5)$$

onde P_r P_g e P_b são os valores das três cores que compõem o *pixel* P. Os pesos que os multiplicam seguem a recomendação da União Internacional de Telecomunicações (ITU) (ITU-R, 2011).

Figura 8. Diferentes posições oculares e suas posições calculadas marcadas em vermelho.

Módulos Auxiliares O controle da SRAM serve como interface por onde os acessos tanto de leitura e escrita da mesma passam, simplificando assim a comunicação com outras partes do *hardware*.

A conversão de endereço garante o acesso fácil às posições de memória usando somente as coordenadas do *pixel*. Essa conversão é explicitada pela equação 6 onde iX e iY são os índices do *pixel* a ser acessado.

$$P_{memoria} = iX + 640 \times iY \qquad (6)$$

Nesse caso 640 é o valor da resolução horizontal do quadro e deve ser mudado de acordo com a resolução da imagem a ser pesquisada.

4. RESULTADOS

Todos os resultados aqui expostos foram obtidos com a mesma configuração da FPGA, o que significa que o tempo de 6 minutos e 48 segundos foi o mesmo para todas as medidas. A Figura 8 mostra as diferentes posições para o olho assim como um ponto vermelho, que é o resultado calculado da posição da pupila. O método se mostrou capaz de encontrar a pupila em variadas disposições oculares. Os pontos brancos que aparecem nos resultados são o reflexo dos LEDs e não interferiu com os resultados.

Em relação aos recursos usados, o compilador Quartus II reportou um uso de 2.292 unidades lógicas, o que corresponde a 7% do total disponível, se mostrando um design muito compacto. Para um outro trabalho disponível na literatura, Kim et al. (2012) mostram um uso de aproximadamente 39.914 unidades lógicas, que para a

FPGA escolhida por eles corresponde a um uso de 76% das unidades lógicas disponíveis.

5. CONCLUSÃO

Na pesquisa apresentada neste trabalho, foi usada uma câmera TRDB-D5M conectada a uma placa de desenvolvimento DE2-115 munida de uma FPGA Cyclone IV. Como proposto por este trabalho, usando uma câmera conectada à uma FPGA é possível encontrar a posição ocular com precisão pelo método de correspondência de modelos. O método, porém, em sua implementação leva um tempo considerável para completar o processo.

Algumas medidas poderiam ser tomadas para diminuir esse tempo como a diminuição da resolução pela metade, o que acarretaria em um processo dezesseis vezes mais rápido. Uma outra maneira de acelerá-lo seria reimplementando essa pesquisa de maneira paralelizada visto que o código atual ocupa somente 7% das unidades lógicas disponíveis na Cyclone IV. Um processo de paralelização que multiplica a implementação atual por dez ainda teria cerca de 30% das unidades lógicas livres para implementação da lógica adicional que governaria as várias instâncias e organizaria seus resultados.

REFERÊNCIAS

Brunelli, R. (2009). *Template Matching Techniques in Computer Vision: Theory and Practice*. John Wiley Sons.

Coburn, J.Q., Freeman, I., and Salmon, J.L. (2017). A Review of the Capabilities of Current Low-Cost Virtual Reality Technology and Its Potential to Enhance the Design Process. *Journal of Computing and Information Science in Engineering*, 17(3). doi:10.1115/1.4036921. URL `https://doi.org/10.1115/1.4036921`. 031013.

Duchowski, A.T. (2009). *Eye tracking methodology: theory and practice*. Springer, 2nd edition.

Guenter, B., Finch, M., Drucker, S., Tan, D., and Snyder, J. (2012). Foveated 3D graphics. *ACM Transactions on Graphics (TOG)*, 31(6), 164.

ITU-R (2011). Studio encoding parameters of digital television for standard 4:3 and wide-screen 16:9 aspect ratios. Recommendation BT.601-7, International Telecommunication Union.

Kim, D.K., Jung, J.H., Nguyen, T.T., Kim, D.J., Kim, M.S., Kwon, K.H., and Jeon, J.W. (2012). An FPGA-based parallel hardware architecture for real-time eye detection. *JSTS:Journal of Semiconductor Technology and Science*, 12(2), 150–161. doi:10.5573/jsts.2012.12.2.150.

Kolb, H. (2011). Simple anatomy of the retina. URL `https://webvision.med.utah.edu/book/part-i-foundations/simple-anatomy-of-the-retina/`.

Latecki, L.J. (2005). Template matching. *Temple University, Lecture Slides*.

Li, X. and Wee, W.G. (2009). An efficient method for eye tracking and eye-gazed FOV estimation. *2009 16th IEEE International Conference on Image Processing (ICIP)*. doi:10.1109/icip.2009.5413997.

Sutherland, I.E. (1968). A head-mounted three dimensional display. In *Proceedings of the December 9-11, 1968, Fall Joint Computer Conference, Part I*, AFIPS '68 (Fall, part I), 757–764. ACM, New York, NY, USA. doi:10.1145/1476589.1476686. URL `http://doi.acm.org/10.1145/1476589.1476686`.

Terasic (2010). DE2-115 user manual. Recommendation Rev. 1.02, Terasic.

Terasic (2017). TRDB-D5M 5 megapixel digital camera development kit user manual. Recommendation Rev. 3, Terasic.

Yan, B., Zhang, X., and Gao, L. (2009). Improvement on pupil positioning algorithm in eye tracking technique. *2009 International Conference on Information Engineering and Computer Science*. doi:10.1109/iciecs.2009.5363013.