Collision.h Comentada

Nos tempos de SDL 1.2 - para os quais nenhum de nós gostaria de voltar - ao girar um Sprite, era possível extrair da imagem rotacionada um Axis-Aligned Bounding Box, e usá-lo na nossa detecção de colisões. Esse AABB era impreciso, mas para os propósitos da disciplina, funcionava, e deixavamos por conta dos alunos implementar a própria função de colisão.

Com a SDL 2.0, essa possibilidade sumiu, e examinando a complexidade de implementar a detecção de colisão do zero, fosse ela baseada em AABB ou OBB, concluímos que seria ruim passar essa tarefa para os alunos. Tratam-se de algoritmos bastante bug prone, e que requerem conhecimentos de Álgebra Linear, assunto com o qual são raros os alunos que se sentem confortáveis.

Optamos por implementar a nossa própria função de colisão e fornecê-la a vocês no Trabalho 6. Você pode encontrá-la na Collision.h, no Moodle. Ela recebe dois Rects e o ângulo de rotação de cada um em **radianos**, e retorna se estão colidindo ou não.

O método usado na função dada é baseado no Separating Axis Theorem, ou SAT. O SAT afirma que se podemos desenhar uma linha separando dois polígonos convexos*, eles não colidem. Enunciado desta forma, o teorema parece bastante trivial, no entanto, achar a tal linha para dois polígonos quaisquer não é tão fácil.

A forma mais técnica de enunciar o SAT seria algo assim: Se pudermos encontrar um eixo onde as projeções dos dois objetos não se intersectam, podemos concluir que os dois objetos não colidem, já que, perpendicular a este eixo, existe um outro eixo que separa os objetos em questão (ver fig. 1 abaixo).

Isso responde a pergunta de como podemos encontrar a linha, mas cria duas novas: não sabemos como projetar um polígono dado num eixo, nem que eixos usar. A resposta da segunda pergunta varia de acordo com os polígonos em uso. Como estamos trabalhando apenas com retângulos, é suficiente testar os quatro eixos paralelos aos lados dos dois retângulos. Ainda assim, precisamos achá-los, então, sem mais delongas, vamos observar, no plano cartesiano, quais as operações realizadas pela IsColliding.

* O SAT também pode ser usado para checar colisões entre um polígono e um círculo, mas não estamos interessados nesse caso aqui. Para o caso de um par de círculos, ele também funcionaria, no entanto, é mais simples checar se a distância entre os centros é menor que a soma dos raios.

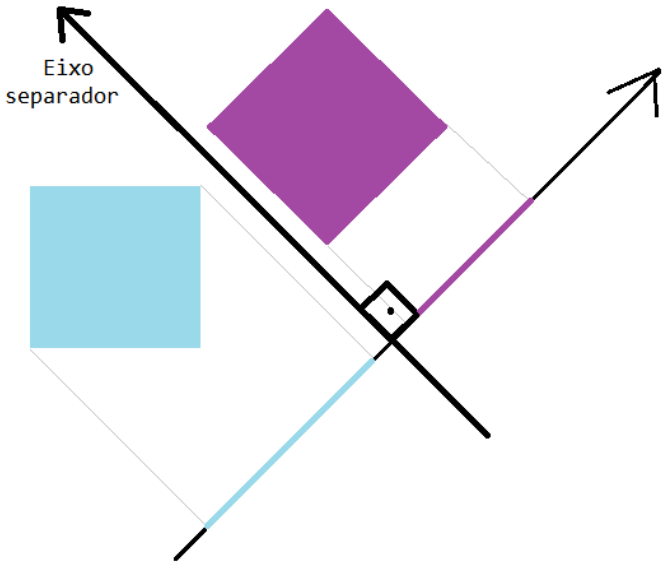
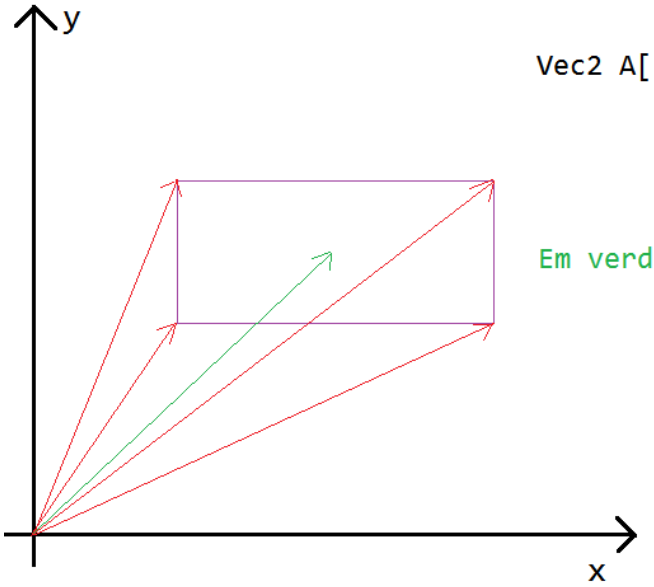


Fig.1 - Eixo separador, encontrado a partir de um eixo onde as projeções não se intersectam

As primeiras linhas da função criam dois arrays de vetores A e B, baseados nas coordenadas dos rects. Esses são os vértices dos retângulos não rotacionados. A primeira coisa que faremos é descobrir o efeito da rotação nesses pontos.




```
Vec2 A[] = { Vec2( a.x, a.y + a.h ),
             Vec2( a.x + a.w, a.y + a.h ),
             Vec2( a.x + a.w, a.y ),
             Vec2( a.x, a.y )
           };

Em verde: a.GetCenter()
```

Fig.2 - Vec2 A[]

Vamos examinar, por partes, a seguinte linha:

```
v = Rotate(v - a.GetCenter(), angleOfA) + a.GetCenter();
```

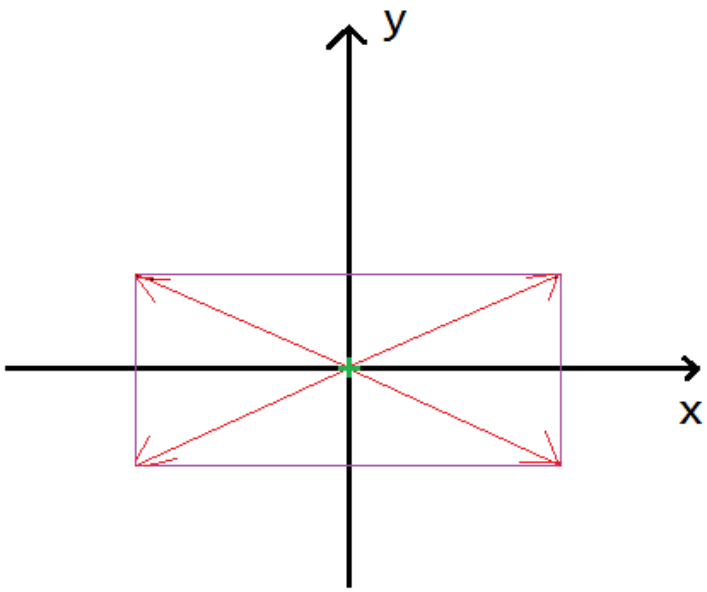


Fig.3 - (v - a.GetCenter())

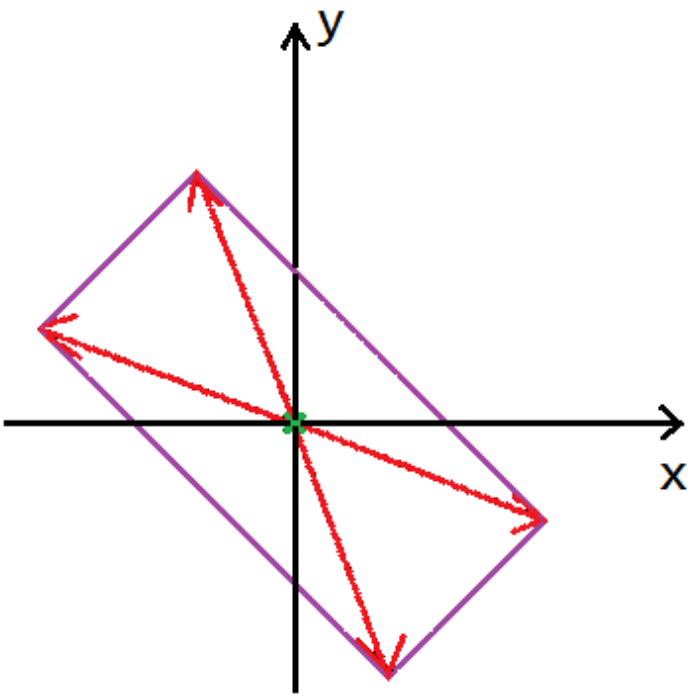


Fig.4 - Rotate(v - a.GetCenter(), angleOfA), com angleOfA igual a pi/4, ou 45°

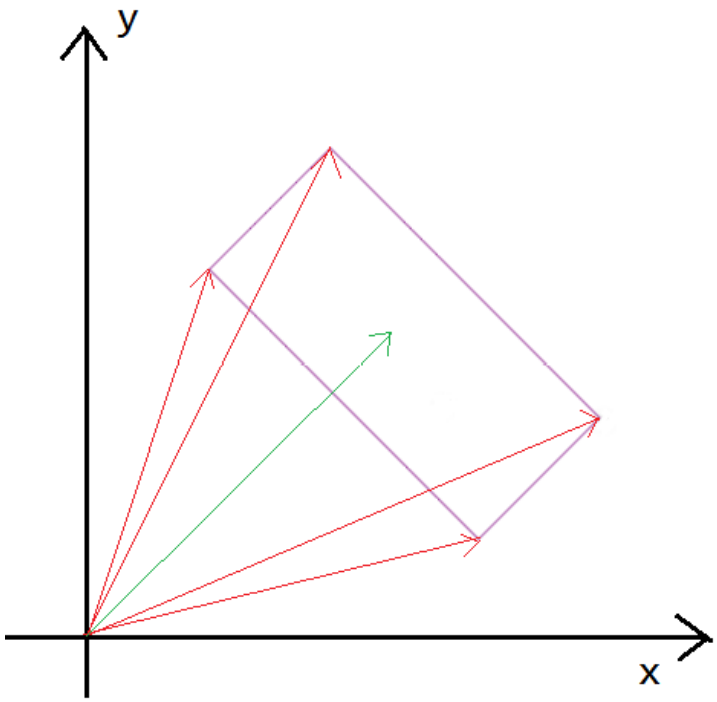


Fig.5 - v = Rotate(v - a.GetCenter(), angleOfA) + a.GetCenter()

Após a execução dos fors sobre A[] e B[], teremos, nestes arrays, os pontos de cada Rect, agora orientados de acordo com a rotação desejada. Agora, aos eixos.

```
Vec2 axes[] = { Norm(A[0] - A[1]), Norm(A[1] - A[2]),
                Norm(B[0] - B[1]), Norm(B[1] - B[2]) };
```

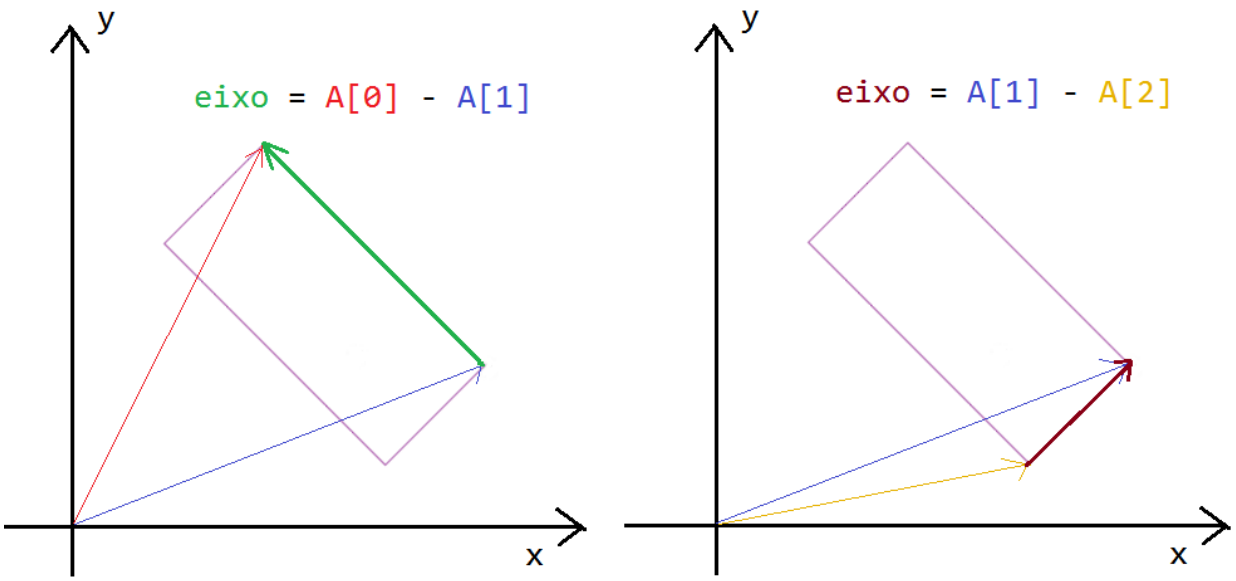


Fig. 6 - Encontrando os lados do retângulo para encontrar os eixos

Uma subtração entre determinados pontos do retângulo nos dá um vetor paralelo ao lado. Normalizando-o, teremos o vetor unitário que representa o nosso eixo a ser testado.

A projeção de um vetor em um outro vetor é dada por:

$$u \cdot v \; / \; |v|$$

Onde · é o produto escalar e |v| é a magnitude de v. Como o nosso eixo é um vetor unitário (magnitude 1), fazemos:

```
for (int i = 0; i < 4; ++i) P[i] = Dot(A[i], axis);
```

Temos, assim, quatro valores que representam os quatro vértices do nosso rect naquele eixo.

```
float minA = *std::min_element(P, P + 4);
float maxA = *std::max_element(P, P + 4);
```

Selecionando o menor e maior valor encontrado, chegamos finalmente à nossa projeção: ela é dada pelo intervalo (minA, maxA) no eixo. Da mesma forma, a projeção de B está em (minB, maxB).

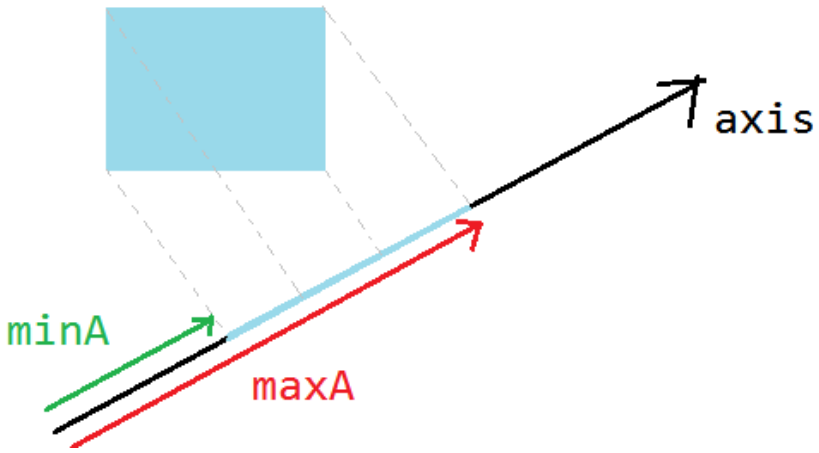


Fig.7 - Retângulo projetado num eixo dado

Podemos finalmente partir para o teste de colisão em si: Se dois intervalos se intersectam, o valor mínimo de um deles é maior que o máximo do outro, ou vice-versa.

```
if (maxA < minB || minA > maxB) return false;
```

Assim que encontrarmos intervalos sem interseções, podemos

interromper os testes: já sabemos que existe um eixo separador para aquele par de retângulos, e isso, de acordo com o SAT, significa que eles não estão colidindo. Esta é uma das vantagens do SAT: menos testes são feitos quando não há colisão, que é o caso mais comum numa cena de jogo qualquer.

Somente se todos os eixos tiverem interseção, podemos afirmar que há colisão. Os casos a seguir mostram resultados possíveis da análise das projeções:

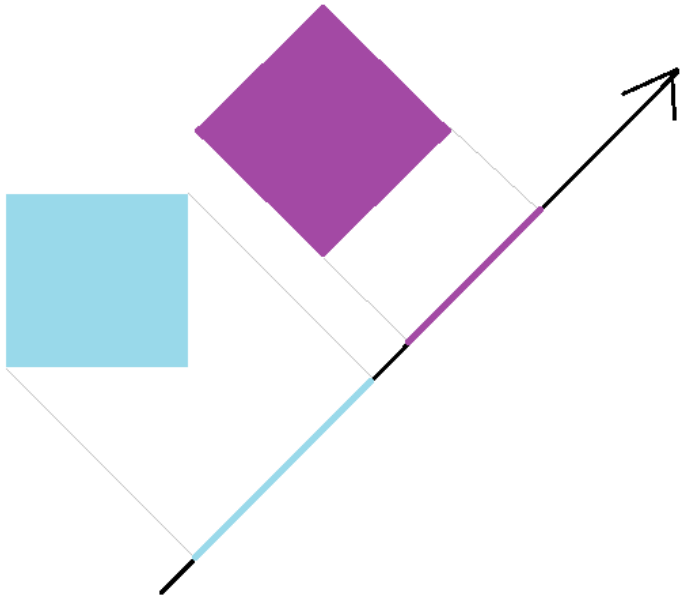


Fig.8 - Caso 1: Os objetos não colidem, e o primeiro eixo testado já nos dá o eixo separador

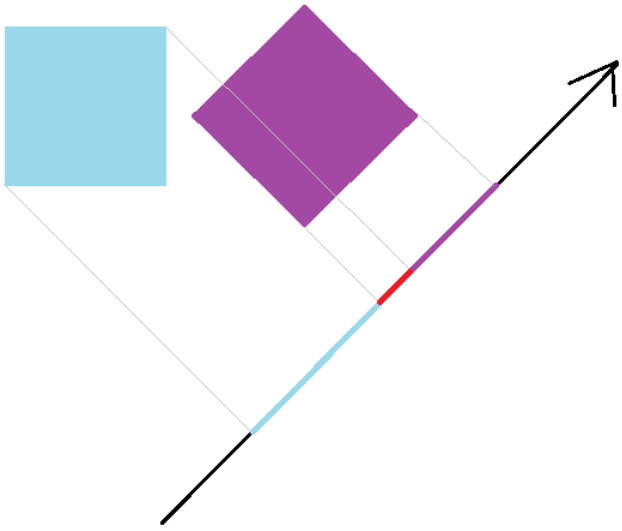


Fig.9 - Caso 2: Os objetos não colidem, mas o primeiro eixo testado mostra intersecção. Testamos os eixos seguintes. O eixo paralelo ao lado inferior do quadrado azul revelará o eixo separador.

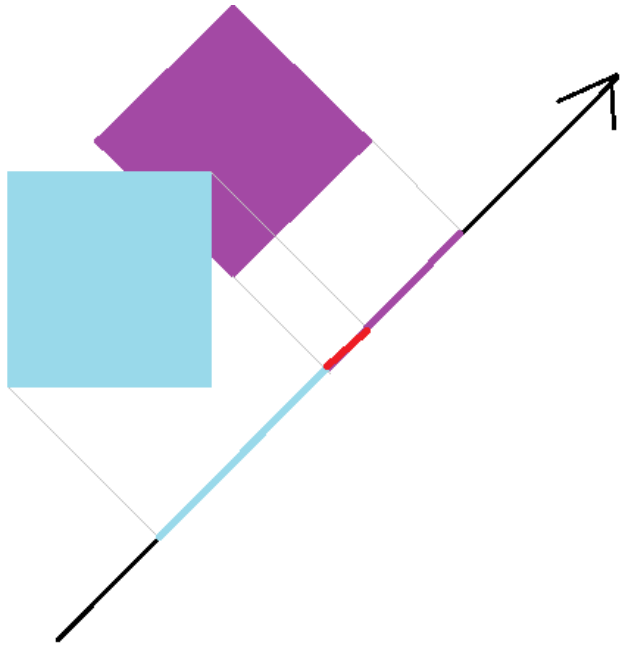


Fig.10 - Os objetos estão colidindo, e para todos os quatro eixos, as projeções vão se intersectar, fazendo com que a função retorne true.

E é assim que a função IsColliding funciona.

Admito que essa explicação pode não ser importante para vários de vocês, dado que para a maioria dos jogos, a colisão com OBBs já basta. Mas acontece de alguns alunos precisarem de colisores com outras formas, por exemplo, então essa explicação fica tanto para os mais inquisitivos, quanto para os que precisam alterar a Collision.h para os seus propósitos.

Qualquer que seja o seu caso, espero que tenha achado a explicação visual interessante.