Sobre Matemáticas y Programación

Walter Mora **F.**
Escuela de Matemática - Instituto Tecnológico de Costa Rica
wmora@itcr.ac.cr

Resumen

Se hace una descripción de cómo se usan las matemáticas en la solución de algunos problemas de graficación por computadora, usando elementos de Algebra Lineal y matemáticas generales. Se presentan las implementaciones en Java de los ejemplos.

Introducción

En este trabajo vamos a abordar algunas ideas matemáticas para resolver algunos problemas de implementación en el contexto de manejo de gráficos por computadora. Muchas de las ideas provienen del Algebra Lineal en general. En esencia se intenta mostrar como algunas tareas de programación pueden convertirse en un campo adecuado para trabajar en 'matemática aplicada'. La solución de problemas de gráficos requiere habilidades de programación y algoritmos eficientes. En los problemas que se presentan, se muestra la parte matemática que resuelve el problema y la solución de algunos problemas de índole puramente computacional. Los mejores algoritmos para las tareas usuales de programación de gráficos se pueden encontrar en los libros dedicados a la graficación por computadora (Computer Graphics). Una bibliografía básica se puede obtener al final de este trabajo.

Algoritmos eficientes

Desde el punto de vista de las matemáticas, muchas veces importa solo la validez teórica de un resultado y no su eficiencia práctica. Pero en el mundo de los cálculos por computadora, importa el número de operaciones para llevar a cabo una tarea.

Por ejemplo, en el contexto de las matrices $A(BC) = (AB)C$. Es decir, da lo mismo hacer el producto $A(BC)$ que el producto $(AB)C$. El resultado es el mismo. Desde el punto de vista de la computación estos cálculos pueden ser muy distintos en términos de esfuerzo computacional. En efecto, si calculamos solamente el número de multiplicaciones que requiere cada una de estos productos de matrices, podemos obtener resultados dramáticamente distintos!

Si tenemos las matrices $A_{m\times n} = (a_{ij})$ y $B_{n\times k} = (b_{ij})$ entonces, si denotamos la fila i-ésima de A con

$F_i = (a_{i1}, \cdots, a_{in})$ y la columna j-ésima de B con $C^j = (b_{1j}, \cdots, b_{nj})$ entonces

$$AB = \sum_{i=1}^{m}\sum_{j=1}^{k} F_i \cdot C^j = \sum_{i=1}^{m}\left(\sum_{j=1}^{k}\left(\sum_{h=1}^{n} a_{ih}b_{hj}\right)\right)$$

de donde, para hacer la multiplicación $A_{\mathrm{m}\times\mathrm{n}}B_{\mathrm{n}\times\mathrm{k}}$ se requieren $m \cdot k \cdot n$ multiplicaciones (no estamos contando sumas).

Así, la multiplicación $A_{10\times100}B_{100\times5}$ requiere 5000 multiplicaciones mientras que $B_{100\times5}C_{5\times50}$ requiere 25000 multiplicaciones, finalmente

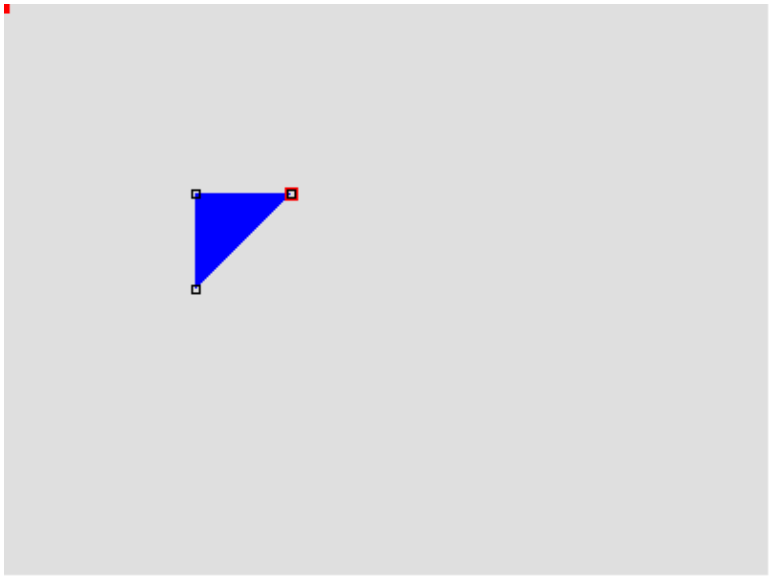
- $(A_{10\times100}B_{100\times5})C_{5\times50}$ requiere 5 000 + 2 500 = 7 500 multiplicaciones
- $A_{10\times100}(B_{100\times5}C_{5\times50})$ requiere 25 000 + 50 000 = 75 000 multiplicaciones

Así, aunque $(AB)C = A(BC)$, computacionalmente es mejor calcular $(AB)C$ en este caso, pues requiere 67 500 multiplicaciones menos!

Una discusión detallada de este tópico se puede ver en [Cormen]

Punto interior de un triángulo

En muchas tareas de programación de gráficos, se necesita seleccionar y arrastrar un objeto geométrico. Para simplificar la discusión, supongamos que tenemos el siguiente problema: tenemos un triángulo (que se puede deformar arrastrando sus vértices) y queremos trasladarlo arrastrando el ratón sobre él. En el "programita" que sigue abajo, se puede arrastrar el triángulo con el ratón y también se puede deformar el triángulo arrastrando los vértices. Cada vez que hacemos clic dentro de la zona gris se despliega un punto rojo, excepto si hacemos clic en el interior del triángulo.



Arrastrar el triángulo. Se puede deformar el triángulo arrastrando sus vértces con el mouse

Para hacer la implementación necesitamos resolver tres problemas

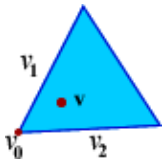
1. Cómo decidir si el 'ratón' está sobre el triángulo?

2. Arrastrar el triángulo

3. Solucionar el problema de refrescamiento de imágenes

Punto interior a un triángulo

Para resolver el problema de decidir si el ratón esta sobre el triángulo, hacemos uso de algunas ideas del Algebra Lineal. Para esto consideremos un ángulo con un vértice v_0, como se muestra en la figura que sigue.



En esta figura, los vectores v_1 y v_2 son los rayos del ángulo con vértice en v_0. Ahora tomamos un punto v y lo expresamos como

$$v = v_0 + a_1 v_1 + a_2 v_2$$

resolviendo para a_1 y a_2 obtenemos

$$a_1 = \frac{v \otimes v_2 - v_0 \otimes v_2}{v_1 \otimes v_2}$$

$$a_2 = \frac{v \otimes v_1 - v_0 \otimes v_1}{v_1 \otimes v_2}$$

donde $u \otimes v := u_x v_y - u_y v_x$

Nota: La fórmula *definida* para vectores en dos dimensiones $u \otimes v$, es una derivación del cálculo formal $u \times v = (u_x, u_y, 0) \times (v_x, v_y, 0) = (u_x v_y - u_y v_x)\mathbf{k}$

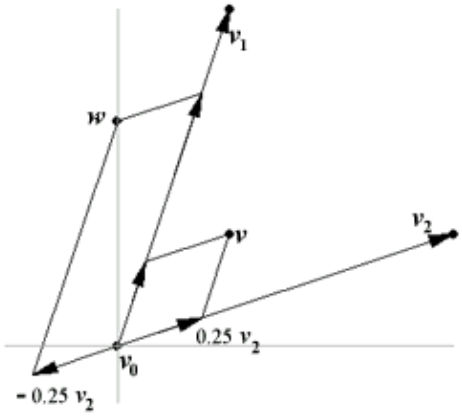
Es claro que si $a_1 > 0$ y si $a_2 > 0$, entonces v estará en el interior del ángulo con vértice en v_0.

De esta manera, v esta en el interior del triángulo si es interior a cada uno de los tres ángulos del triángulo.

Por ejemplo, consideremos $v0 = (0,0)$, $v1 = (1,3)$ y $v2 = (3,1)$.

Si tomamos $v = (1,1)$ entonces $v = v0 + 0.25\, v1 + 0.25\, v2$, así que como $a1 = 0.25 > 0$ y $a2 = 0.25 > 0$ entonces v está en el interior del ángulo

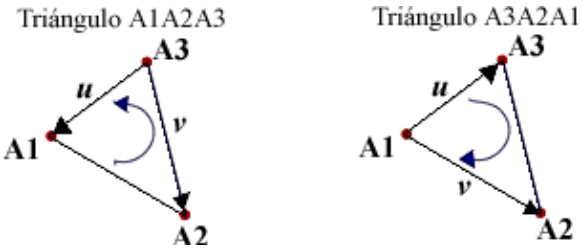
Si tomamos $w = (0,2)$ entonces $w = v0 + 0.75\, v1 - 0.25\, v2$, así que como $a1 = 0.75 > 0$ y $a2 = -0.25 < 0$ entonces w está fuera del interior del ángulo



Este resultado se puede reformular en términos de *orientación* de un triángulo. Este otro punto de vista nos permite formular un algoritmo sencillo para resolver el primer problema.

Orientación de tres puntos

Decimos que tres puntos del plano tienen orientación nula si son colineales, en otro caso decimos que tres puntos A1, A2 ,A3 están orientados de manera positiva si para visitar estos tres puntos en el *orden dado*, nos movemos contra-reloj y que están orientados de manera negativa si para visitar estos tres puntos en el *orden dado*, nos movemos a favor del reloj.



La orientación del triángulo depende del orden en que se especifican
sus vértices, usando el producto cruz $u \times v$ podemos *calcular* la orientación

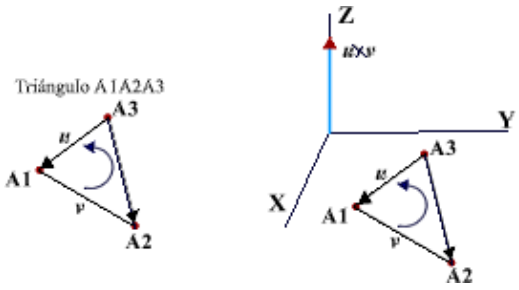
Con el producto cruz definido en $\mathbf{R}^3$ se puede establecer una manera sencilla para determinar la orientación de tres puntos *A1*, *A2*, *A3 (en este orden)*. LLamemos $u = A1 - A3$ y $v = A2 - A3$ entonces

$$u \times v = (u_x, u_y, 0) \times (v_x, v_y, 0) = (u_x v_y - u_y v_x)\mathbf{k}$$

La fórmula que resultó es natural ya que los tres puntos están en el plano XY, por lo tanto el producto cruz es un vector en la dirección del eje Z. Podemos usar el signo de la componente $u_x v_y - u_y v_x$ para definir la orientación, así si

$$u \otimes v := u_x v_y - u_y v_x$$

entonces, la orientación es positiva si $u \otimes v > 0$ y la orientación es negativa si $u \otimes v < 0$



Por ejemplo si A1 = (2,3), A2 = (3,5), A3 = (1,4) entonces

Si $u = A1 - A3$ y $v = A2 - A3$ *tendríamos* $u \otimes v = 3 > 0$, *por tanto la orientación de A1A2A3 es positiva*

Si $u = A3 - A1$ y $v = A2 - A1$ *tendríamos* $u \otimes v = -3 < 0$, *por tanto la orientación de A3A2A1 es negativa*

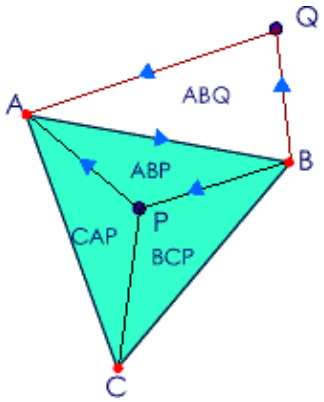
Finalmente, decimos que el triángulo $\triangle ABC$ esta orientado positivamente si los vértices A, B, C están, en este orden, orientados positivamente. De manera análoga decimos que el triángulo está orientado negativamente si los vértices A, B, C están, en este orden, orientados negativamente. Así, en el ejemplo anterior, el $\triangle A3A2A1$ está orientado negativamente.

Punto interior en un triángulo y orientación

Se puede reformular el resultado acerca de los puntos interiores en un triángulo de esta manera

Teorema. *Consideremos un triángulo* $\triangle ABC$ *y un punto P del plano. P está en el interior de este triángulo si la orientación de los triángulos* $\triangle ABP$, $\triangle BCP$ *y* $\triangle CAP$ *es la misma que la orientación del triángulo* $\triangle ABC$.

Aquí hay que recordar que la orientación de cada triángulo se determina de acuerdo a la dirección del movimiento cuando se visitan los vértices en el orden especificado. Consideremos el triángulo ABC de la figura. El punto Q no está en el triángulo. Esto se puede verificar computacionalmente observando que el triángulo ABQ tiene orientación contraria al triángulo ABC. Los triángulos ABP, CAP,BCP si mantienen la orientación así que se puede establecer computacionalmente que P está en el interior del triángulo.



B no es interior al triángulo ABC pues el triángulo ABQ tiene orientación contraria al triángulo ABC

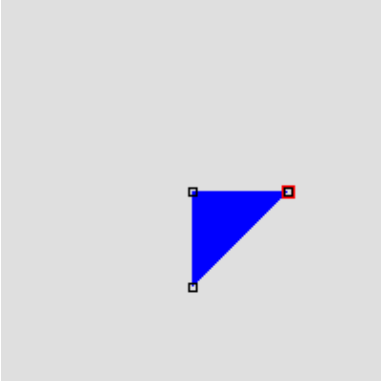
En este teorema, la orientación de un triángulo es la misma que la orientación de sus tres vértices, así que podemos establecer un algoritmo sencillo para decidir si un punto está o no en el interior de un triángulo. En efecto, consideremos un triángulo $\triangle A1A2A3$. Llamemos $u = A1 - A3$ y $v = A2 - A3$, entonces el signo de $u \otimes v$ decide la orientación del triángulo. En el fragmento de código que sigue, a $u \otimes v$ se le llama "areaT" pues

con esta fórmula también se puede calcular el área del triángulo (en realidad con la norma del producto cruz)

```
    //código Java
float areaT(Point2D A1,Point2D A2,Point2D A3)
   {//área del triángulo  A1A2A3, es > 0 sii esta orientado positivamente,else < 0
    return (A1.x - A3.x)*(A2.y - A3.y)-(A1.y - A3.y)*(A2.x - A3.x);

   }//

boolean estaEnTri(Point2D A1,Point2D A2,Point2D A3, Point2D P)
   {   // Decide si un punto P está dentro del triángulo orientado A1A2A3

       if(areaT(A1,A2,A3)>=0)
              return areaT(A1, A2, P) >= 0 &&
                     areaT(A2, A3, P) >= 0 &&
                     areaT(A3, A1, P) >= 0;
       else   return areaT(A1, A2, P) <= 0 &&
                     areaT(A2, A3, P) <= 0 &&
                     areaT(A3, A1, P) <= 0;

   }//
```

Observemos que permitimos que `AreaT(Ai,Aj,P)` sea 0. Con esto dejamos que los puntos de los lados sean también considerados interiores.

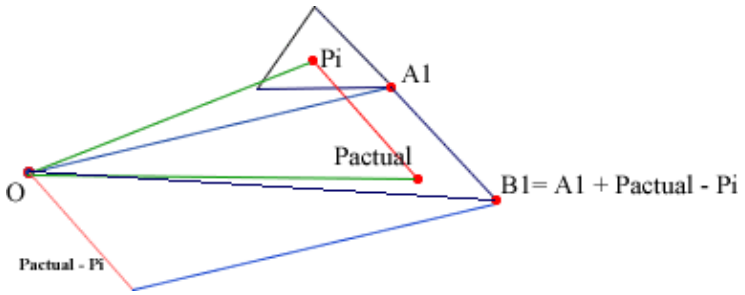
En el siguiente "programita", se muestra como trabaja el algoritmo. Al entrar el ratón al triángulo, se muestra un punto amarillo (no hay necesidad de arrastrar el ratón, solo moverlo). El triángulo se puede deformar.



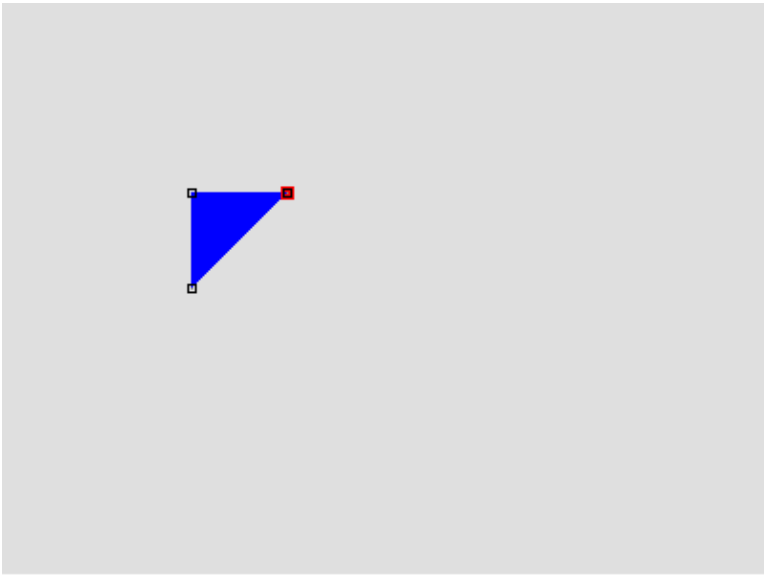
En la práctica estos métodos trabajan bien si el próposito es decidir si un punto es interior a un triángulo dado pero eventualmente, con un valor flotante x, podemos considerar reemplazar un test de la forma $x \geq 0$ con $x \geq -\epsilon$ donde se podrá poner, por ejemplo $\epsilon = 10^{-6}$

Arrastrar el triángulo

Para arrastrar el triángulo $\triangle A1A2A3$ debemos hacer una traslación de los vértices en la dirección del movimiento del ratón. Si llamamos *Pini* al punto inicial del movimiento (presionamos el botón derecho y arrastramos) y *Pactual* a cada nuevo punto en el recorrido del ratón mientras es arrastrado, entonces el triángulo $\triangle A1A2A3$ se debe actualizar como $\triangle B1B2B3$ con $Bi = Ai + Pactual - Pini$ como se observa en la figura que sigue.

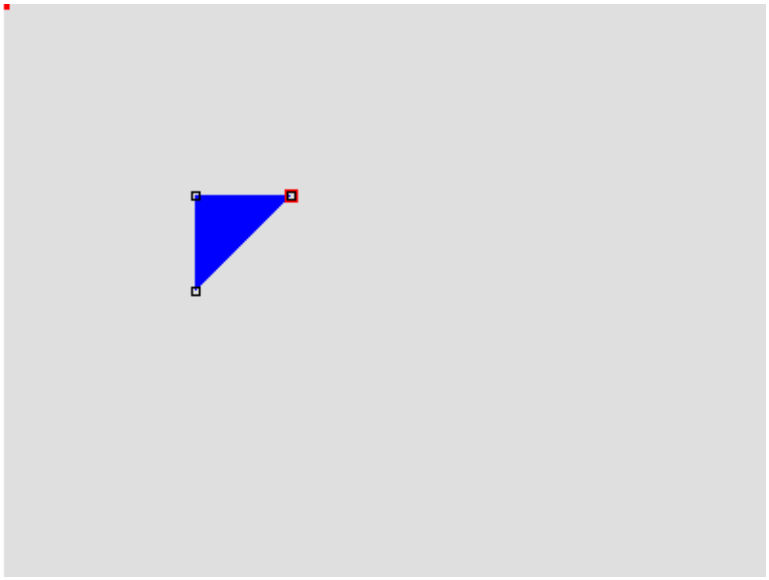


Podemos visualizar esta traslación arrastrando el triángulo en el siguiente "programita". Al arrastrar el ratón se muestra el vector que da la dirección de traslación. Si lo arrastramos sobre el triángulo, se muestra la traslación de los vértices.



Refrescamiento de imágenes

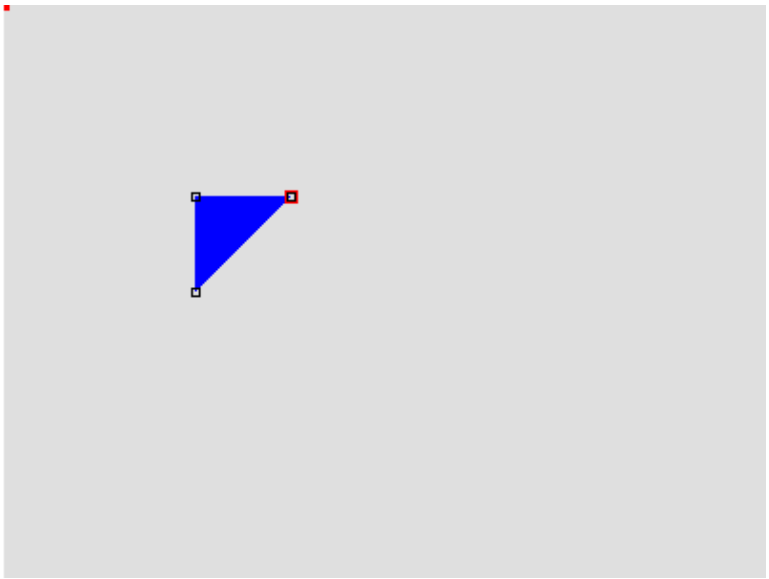
Como se puede comprobar en el "programita" que está más abajo, al arrastrar el triángulo se puede notar un "parpadeo". Esto se debe a que todos los pixeles de la imagen no se exhiben de manera simultánea.



Para evitar este problema, se exhibe primero una imagen mientras se construye la otra imagen 'fuera de pantalla' (es decir, no se despliega la imagen hasta que todos los cálculos hayan sido hechos), luego cuando llega el momento de exhibir la nueva imagen, se borra la imagen anterior desplegando un triángulo en blanco y luego se despliega la nueva imagen. Esta técnica se conoce como 'doble buffer' de gráficos. Esta fue la técnica que se uso en los "programitas" anteriores.

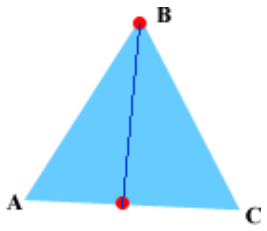
Alturas de un triángulo

Consideremos el problema de hacer un programa que despliega un triángulo (que se puede deformar arrastrando sus vértices) de tal manera que si arrastramos cualquiera de sus vértices, se despliega la altura correspondiente. Por ejemplo, arrastre cualquiera de los vértices del triángulo en el "programita" que sigue

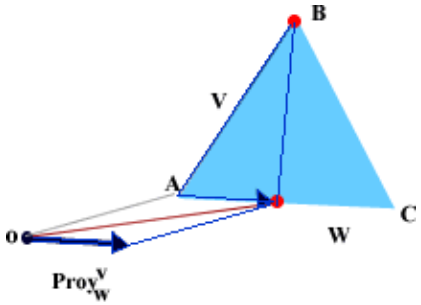


Para implementar este programa necesitamos una manera eficiente para calcular el punto en el lado opuesto al vértice, donde cae la altura. Sin duda una manera natural es usar proyecciones. Si se conoce la orientación del triángulo también se puede decidir rápidamente si hay que extender o no un lado y la dirección.

Consideremos una triágulo $\triangle ABC$ en la figura que sigue. Supongamos que queremos tirar la altura desde el vértice en B hasta el lado opuesto.



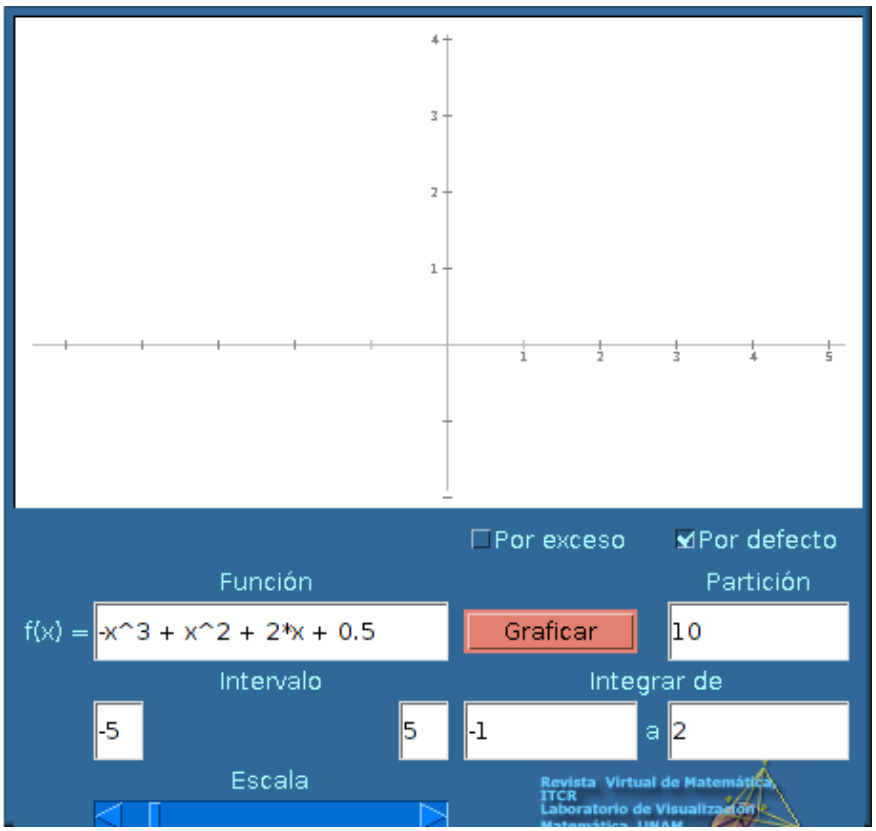
Si llamamos $v = B - A$ y $w = C - A$ entonces la altura desde el vértice en B debe ser el segmento que va desde el vértice en B hasta el punto $P=Proy_{\mathrm{W}}^{\mathrm{V}} + A$ como se ve en la figura que sigue



Puesto que $Proy_{\mathrm{W}}^{\mathrm{V}} = \lambda\, w$ es claro que la altura cae en el segmento CA solo si $0 \leq \lambda \leq 1$. Como $w = C - A$ entonces si $\lambda > 1$, se debe extender el segmento AC a la derecha de C y si $\lambda < 0$ se debe extender a la izquierda de A.

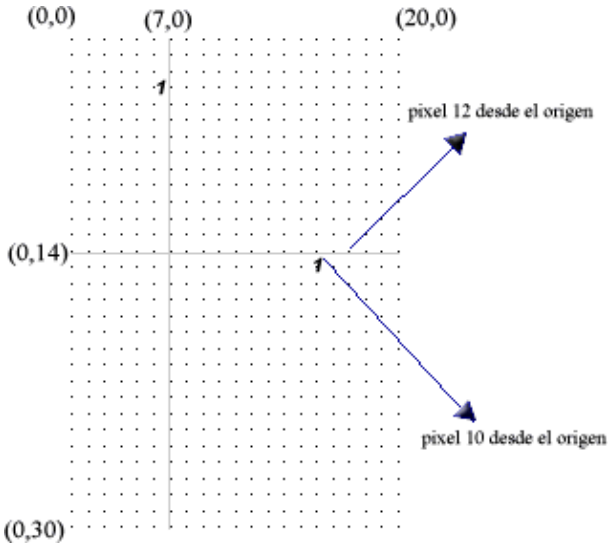
Graficación de funciones. Integral de Riemann

En esta sección vamos a discutir los aspectos matemáticos de la implementación de un graficador, usado para hacer experimentos relacionados con la integral de Riemann. Para empezar teneieno una idea, haga clic en el botón "Graficar" y luego manipule la barra de escala y el intervalo de graficación.



Supongamos que queremos graficar una función $f(x)$ desde $x = a$ hasta $x = b$ en un sistema de coordenadas en el que el eje X tiene escala '1:escalaX', es decir, una unidad son 'escalaX' pixeles y el eje Y tiene escala '1:escalaY'. Vamos a suponer que el eje X va desde *xmin* hasta *xmax*. Y que estos valores son definidos por el usuario.

Primero vamos a establecer la relación entre las coordenadas reales de un punto y sus correspondientes coordenadas en la pantalla de un monitor. La pantalla de un monitor la debemos imaginar como una rejilla finita de puntos, como se ve en la figura que sigue (aquí los pixeles están distanciados de una manera exagerada para ayudar a visualizar la situación). La esquina superior izquierda inicia en (0,0), es decir la fila 0-columna 0.



En este gráfico se muestra una simulación de los pixeles en pantalla.

En la simulación de los pixeles en pantalla del gráfico de arriba, se puede observar que se ha escogido el punto (7,14) como origen de coordenadas, así que respecto a este sistema de coordenadas

- el punto de pantalla (6, 14) corresponde a (-1, 0) respecto al sistema de coordenadas
- el punto de pantalla (17, 14) corresponde a (1, 0) respecto al sistema de coordenadas
- el punto (1,1) respecto al sistema de coordenadas corresponde al punto (17, 4) en pantalla

En general, el punto (p, q) respecto al sistema de coordenadas con origen (x0, y0) corresponde al punto en pantalla
(x0 + p, y0 - q).

Para graficar una función, desde $x = a$ hasta $x = b$, se debe tener en cuenta la escala en el eje *X*.

Si por ejemplo la escala es 1 : 10 (es decir, 1 unidad =10 pixeles), entonces para graficar $f(x)$ desde $x = 0$ a $x = 2$ disponemos solamente de 20 pixeles.

Si queremos graficar $f(x)$ desde $x = a$ hasta $x = b$, podemos unir con segmentos, como máximo, $(b - a)*escalaX$ pares ordenados pues este es exactamente el número de pixeles que tenemos en el eje X entre $x = a$ y $x = b$. Las coordenadas reales de estos pares son (Rxi, Ryi) con

$$Rxi = a + i/escalaX; \quad Ryi = f(Rxi), \qquad i = 0, 2,...,(b - a)*escalaX$$

Por ejemplo, si escalaX= 10 entonces $Rx2 = a + 2/10$, es decir, avanzamos en saltos de una décima. Como la escala es "1:10", cada décima representa un salto de un pixel. No podemos avanzar, con esta escala, en saltos de menos de una décima pues al redondear obtendriamos "0 pixeles" de avance

Las coordenadas correspondientes *en pantalla* son

$$Cxi = Math.round\,(a*escalaX + i); \quad Cyi = Math.round\,(escalaY*f(Rxi)) \quad i = 0, 2,...,(b - a)*escalaX$$

Por ejemplo, si escalaY = escalaX = 10 y $a = 2$ entonces

si $Rx_7 = 2 + 7/10 = 2.7$ entonces $Cx_7 = 27$ pixeles desde el origen de coordenadas

si $Ryi = 1.3454$ entonces $Cyi = 13$ pixeles arriba del origen.

Entonces, si el origen está en (x_0, y_0) (en la pantalla) entonces el gráfico se obtiene uniendo los pares

$$(x_0 + Cxi,\ y_0 - Cyi); \qquad i = 0, 2,...,(b - a)*escalaX$$

Con estos cálculos podemos usar exactamente los pixeles disponibles (o menos, si es el caso). El número de pares ordenados puede aumentar o disminuir de manera automática al variar la escala en el eje *X* y el eje *Y,* es decir, si queremos evaluar valores de *x* más cercanos, podemos aumentar la escala en X. Muchas veces, por problemas de redondeo se debe aumentar también la escala en Y.

Los aumentos en la escala son necesarios para capturar el comportamiento de funciones que oscilan muy rápido en pequeños intervalos (como y = sen (1/x))

Para dibujar *n* rectángulos entre la curva de ecuación $y = f(x)$ y el eje *X* entre $x = a$ y $x = b$ se debe tomar en cuenta la 'escalaX' pues está determina el número máximo de rectángulos que podemos dibujar.

Si decidimos dibujar *n* rectángulos entre $x = a$ y $x = b$ entonces $Rxi = a + i*(b - a)/n$, pero esto tiene sentido a la hora de pasarlo a coordenadas de pantalla solo si la escala es lo suficientemente grande como para que cada paso
$i*(b - a)/n$ corresponda a 1 o más pixeles en coordenadas de pantalla.

Como es natural, cualquier descuido en los cálculos haría que no aparezcan algunos rectángulos o algunos puntos de la gráfica.

Conclusión

En los ejemplos presentados, hemos abordado las matemáticas necesarias para resolver algunos problemas computacionales de graficación. En estos se muestra como en problemas de graficación por computadora, el problema no es solo tener una solución matemática del problema, sino tener una solución eficiente. Al mismo tiempo, la solución de estos problemas plantean retos que ayudan a `saborear' un poco la matemática aplicada. Finalmente, aunque la programación es una actividad muy creativa, muchas veces se requiere el estudio de algoritmos ya establecidos para obtener una solución eficiente a un problema.

Bibliografía

1. Cormen, T.(editor) *Introduction to Algorithms*. MIT Press. Segunda Edición. 2001.

2. Noble, D. Daniel, J. *Algebra Lineal Aplicada*. Prentice-Hall. Tercera Edición. 1989.

3. Schildt, H. *Java 2*. McGarw-Hill. 4ta Edición. 2001

4. Glassner, A. (Editor) *Graphics Gems*. Morgan Kaufmann. 1990.

5. Arvo, J. (Editor) *Graphics Gems II*. Morgan Kaufmann. 1991.

6. Mortenson, M. *Mathematics for Computers Graphics Applications*. Industrial Press. 1999