INTERSECCIONES

El cálculo de intersecciones en el espacio tiene varias finalidades en CG; podemos citar, a modo de ejemplos, el *clipping* (recorte con planos), el *ray tracing* (reflejo de rayos) y la detección de colisiones.

Se pretende que los algoritmos empleados sean **rápidos, precisos y robustos**. La falta de robustez hace que los errores salten inmediatamente a la vista o peor aún, que se "cuelgue" el programa. Siempre es preferible sacrificar la precisión en aras de la robustez; buscando que el código entregue siempre un resultado útil, en cualquier caso y, si es necesario, se admitirá un pequeño margen de error; por ejemplo: reemplazando un indeseable cero mediante un número muy pequeño.

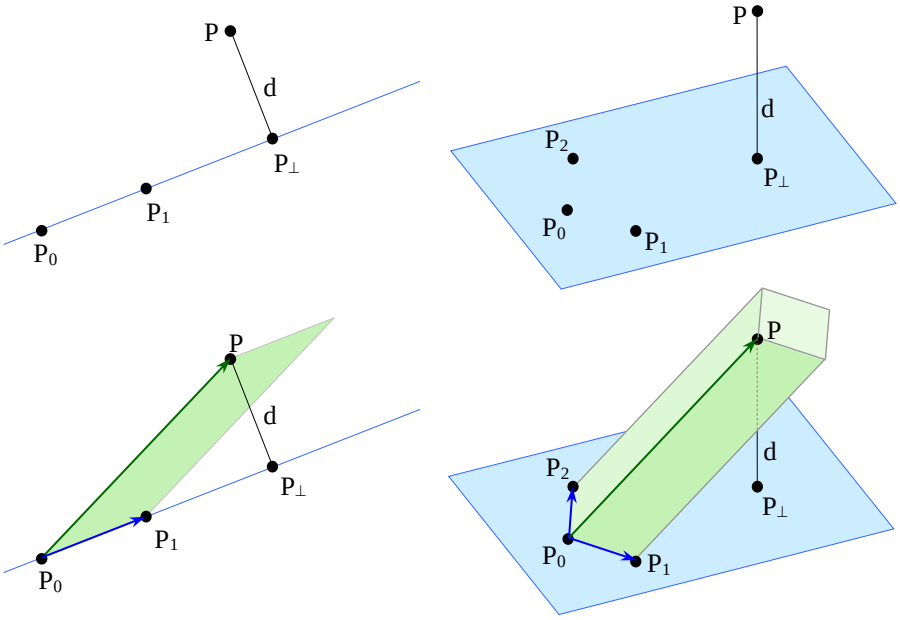
Con las modernas GPU que procesan en el orden de 10^8 triángulos por segundo; a 25 cuadros/seg (fps) se pueden procesar unos 4 millones de polígonos por escena. El requerimiento de velocidad se vuelve obvio, aún sin utilizar la GPU; un *ray–tracing*, por ejemplo, debe analizar intersecciones entre un millón de rayos (WxH ~1000x1000) y miles de polígonos de la escena; la primera respuesta que hay que dar, y muy rápidamente, es si puede o no existir la intersección y con un gran grupo, no con cada polígono.

Veremos algunos pocos ejemplos de cálculo de intersecciones y de técnicas de partición del espacio que nos permitan acelerar estas consultas. La bibliografía sugerida es el libro de Schneider y Eberly.

En CG, los planos se representan mediante un vector homogéneo $\Pi = \{a,b,c,d\}$. En la GPU, la distancia a un punto $P = \{wx,wy,wz,w\}$ se calcula mediante un simple producto escalar: $D = \Pi \cdot P/w|\mathbf{n}|$. Los detalles están en un apéndice de la teoría de transformaciones. Aquí analizaremos los conceptos desde cero.

Pertenencia y Distancia de Punto a Recta y Plano

La distancia entre dos objetos es la **menor** de entre las distancias de sus puntos. De un punto P a un plano o a una recta, es la distancia de P al pie de la perpendicular $P_\perp$ desde P. Esto es así, porque cualquier otro punto P' de la recta o plano, formaría un triángulo rectángulo con P y $P_\perp$; pero cualquier hipotenusa P-P' es mayor que el cateto P-$P_\perp$ que, siendo el mínimo segmento, es el que define la distancia.



En la recta definida por expansión afín de P_0 y P_1, notemos que el paralelogramo que generan los vectores $P_1–P_0$ y $P–P_0$ tiene altura d, que es la distancia de P a la recta y se puede calcular como el área del paralelogramo (base × altura) dividida por la longitud de la base, que es el módulo del vector $P_1–P_0$.

$$\mathbf{d}(P,\{P_0,P_1\}) = \frac{(P-P_0)\times(P_1-P_0)}{\|P_1-P_0\|} = (P-P_0) \times \frac{(P_1-P_0)}{\|P_1-P_0\|}$$

Versor **t**, dirección de la recta

A la derecha quedó un versor tangente **t** normalizado que define la dirección de la recta.

La distancia, del punto P, al plano por tres puntos $\{P_0, P_1, P_2\}$, será el volumen (producto triple) del paralelepípedo (área de la base × altura) dividido por el área de la base (módulo del producto vectorial):

$$d(P,\{P_0,P_1,P_2\}) = \frac{(P-P_0)\cdot((P_1-P_0)\times(P_2-P_0))}{\|(P_1-P_0)\times(P_2-P_0)\|} = (P-P_0) \cdot \frac{(P_1-P_0)\times(P_2-P_0)}{\|(P_1-P_0)\times(P_2-P_0)\|}$$

Versor **n**, normal al plano

El versor de la derecha es el normal (**n**) unitario del plano.

En ambos casos, si el resultado es nulo, el punto pertenece a la recta o al plano.

Una "distancia" es un real sin signo; por lo tanto, las ecuaciones de arriba tienen datos demás. El resultado de la primera ecuación es un vector; es normal al plano definido por la recta y el punto y su módulo es la distancia buscada; la dirección del vector nos indica un punto de vista desde el cual el punto está a la derecha de la recta en ese plano. Para la otra ecuación, la distancia calculada al plano es un escalar con signo y el signo indica de qué lado del plano está el punto: igual u opuesto a la normal.

Si la recta o el plano viene definido mediante otra ecuación (implícita, por ejemplo) se puede calcular la distancia con similar facilidad o llevarla al caso planteado. Si vienen dados por punto y tangente o punto y normal, usamos esos vectores, normalizados, en las ecuaciones anteriores.

Una recta es el conjunto de puntos $P = P_0+\alpha\mathbf{t}$. El vector que define la dirección, puede venir dado o calcularse como la diferencia de dos puntos dados $\mathbf{\Delta}P = P_1-P_0$. Se normaliza sólo si es necesario. Si α es cualquier real, se trata de una recta; si $\alpha \geq 0$ de una semirrecta y si $0 \leq \alpha \leq 1$, de un segmento.

Lo mismo sucede para un plano $\{P_0, \mathbf{n}\}$ o $ax+by+cz+d=0$ o un triángulo $\{P_0, P_1, P_2\}$. Si no viene dada, se calcula: $\mathbf{n} = (P_1- P_0) \times (P_2-P_0)$ o $\mathbf{n} = \{a, b, c\}$, normalizando sólo si es necesario. Para obtener un punto P_0 de la ecuación implícita $\{a,b,c,d\}$, se puede usar $\{-d/a,0,0\}$, $\{0,-d/b,0\}$, $\{0,0,-d/c\}$ o $-d\mathbf{n}/n^2$.

El pie de la perpendicular se puede obtener proyectando $P-P_0$ en la recta o el plano:

$$(P_\perp-P_0) = [(P-P_0)\cdot\mathbf{t}]\ \mathbf{t} / t^2 \qquad (P_\perp-P_0) = (P-P_0) - [(P-P_0)\cdot\mathbf{n}]\ \mathbf{n} / n^2$$

Las divisiones por el cuadrado del módulo se explicitaron por si no vienen normalizados. La primera ecuación es la proyección de $P-P_0$ sobre la recta. En la segunda ecuación, al vector $P-P_0$ se le quita la componente normal: $[(P-P_0)\cdot\mathbf{n}]\ \mathbf{n} / n^2$, quedando sólo la componente en el plano.

En caso de que se desee calcular sólo la pertenencia o no al plano o recta, o de qué lado está el punto, se pueden ahorrar cuentas; a veces no importa la distancia, sino sólo saber si es nula, positiva o negativa. En general, hay que evitar raíces y divisiones para ganar velocidad y robustez. Por ejemplo: para comparar distancias no hace falta dividir siempre por el mismo denominador; los cuadrados, inversas (no infinitas) o cualquier otra función monótona sirve para comparar.

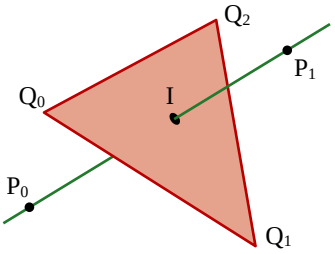
Intersecciones

Sólo haremos un par de casos simples y útiles para CG; el resto se puede encontrar en la bibliografía.

Debe comprenderse que hay muchos métodos para calcular las mismas intersecciones, aquí usaremos las herramientas que ya manejamos. Normalmente, se deben resolver sistemas de ecuaciones, puesto que las intersecciones son los puntos que cumplen las definiciones analíticas de los dos elementos que hay que interceptar. En lugar de plantear sistemas, tratemos de deducir soluciones cerradas, es decir fórmulas y algoritmos que nos den directamente la solución, utilizando los conceptos que acabamos de ver.

Intersección de Segmento y Triángulo

La situación más usual (*clipping*, *ray tracing*) y sencilla es la intersección de un segmento, recta o rayo contra un triángulo o un plano en el espacio.



La plantearemos como el punto de la recta que anula la distancia al plano:

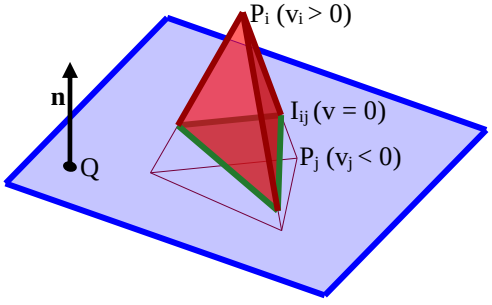
$$(P-Q_0)\cdot\mathbf{n}/|\mathbf{n}| = 0 \ \Rightarrow\ (P-Q_0)\cdot\mathbf{n} = 0 = [P_0+\alpha(P_1-P_0)-Q_0]\cdot\mathbf{n}$$
$$\alpha = (Q_0-P_0)\cdot\mathbf{n} \,/\, (P_1-P_0)\cdot\mathbf{n} \ \Rightarrow\ I = P_0+\alpha(P_1-P_0)$$

El $|\mathbf{n}|$ no hizo falta, se anulará siempre la pseudo-distancia $|\mathbf{n}|d = (P-Q_0)\cdot\mathbf{n}$.

El parámetro α calculado, permite ubicar la intersección en el segmento, recta o rayo (semirrecta). Si hay que verificar que esté en el interior del triángulo, se calculan dos coordenadas baricéntricas de I.

Cuando se deben analizar intersecciones entre un único plano (*clipping*) y muchos segmentos, triángulos o modelos *b-rep* se asigna la pseudo-distancia a cada vértice:

$$v_i = |\mathbf{n}|d_i = (P_i-Q)\cdot\mathbf{n} = P_i\cdot\mathbf{n} - Q\cdot\mathbf{n} \qquad (Q\cdot\mathbf{n} \text{ cte. } \forall i)$$



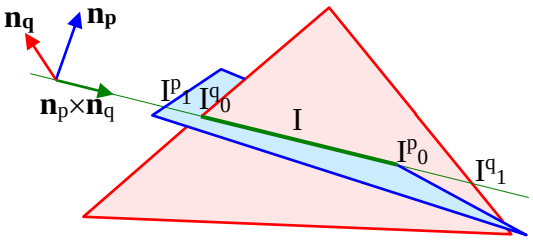
Luego se busca el valor 0, pero sólo en las aristas $\{P_i,P_j\}$ cuyos extremos tengan valores de distinto signo ($v_i\ v_j < 0$):

$$\frac{I_{ij}-P_j}{0-v_j} = \frac{P_i-P_j}{v_i-v_j} \qquad \Rightarrow \qquad I_{ij} = P_j + \underbrace{\frac{v_j}{v_j-v_i}}_{\alpha}(P_i-P_j)$$

Por cuestiones de robustez, hay que analizar qué hacer en caso de que v_j y v_i sean "muy pequeños" (?).

Intersección de Triángulos

Cuando dos triángulos no se superponen en el mismo plano, su intersección, si existe, es un segmento en la recta de intersección de sus planos. Para encontrarla, se deben hallar y ordenar los puntos de intersección de las tres aristas de cada triángulo con el plano del otro triángulo.



Calculamos primero las normales $\mathbf{n}_p$ y $\mathbf{n}_q$ de los planos, como productos vectoriales de dos aristas en cada triángulo.

La intersección pertenece a ambos planos; su dirección es perpendicular a ambas normales: $\mathbf{n}_p\times\mathbf{n}_q$. Si el producto es nulo, los planos son paralelos y no hay intersección, o son coincidentes y no se busca así.

Para obtener los puntos de intersección (aristas vs. planos) se asigna la pseudo-distancia de cada vértice P_i de un triángulo, al plano de los otros tres Q_i: $(P_i-Q_0)\cdot\mathbf{n}_q$; luego se buscan las intersecciones I^p_i, en cada arista cuyos vértices tengan distinto signo. Lo mismo se hace para las aristas de Q con el plano de los P.

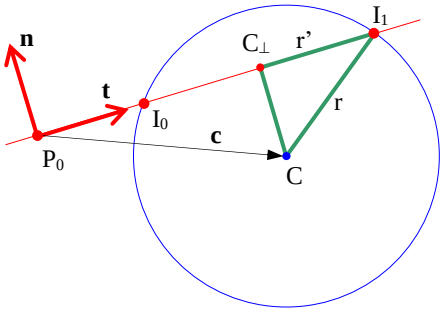
Cada triangulo puede generar hasta dos intersecciones con el otro plano; formando un segmento, que aún puede no interceptar al otro triángulo. La intersección, si existe, es la fracción solapada de esos dos segmentos. Para encontrarla se pueden calcular las coordenadas baricéntricas 1D de los cuatro puntos de intersección en el segmento $\{I^p_0,I^p_1\}$: $\alpha^p_0=0$, $\alpha^p_1=1$, $\alpha^q_{0/1} = (I^q_{0/1}-I^p_0)\cdot(I^p_1-I^p_0)/(I^p_1-I^p_0)^2$. la intersección queda definida por los α extremos del intervalo de intersección real $[0, 1]\cap[\min(\alpha^q_i), \max(\alpha^q_i)]$.

Para garantizar la robustez, habrá que analizar casos particulares y evitar errores de precisión numérica.

Círculos contra Círculos en 2D; en 3D: Esferas o Cilindros contra Esferas o Cilindros.

Sus interiores se interceptan si la distancia entre centros/ejes es menor que la suma de radios. Sólo nos interesa este caso para detectar colisiones; no suele ser necesario calcular la intersección.

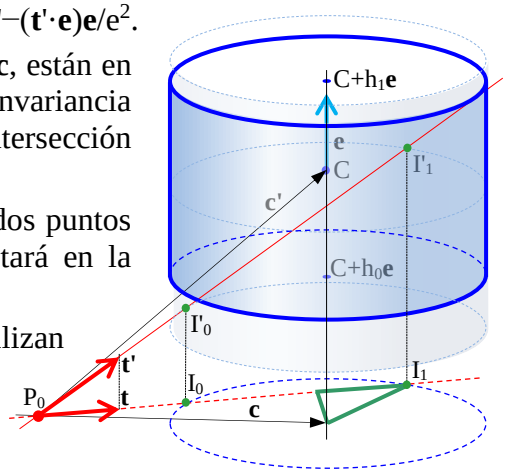
Esferas o Cilindros contra Rectas o Planos



Todos esos casos se pueden tratar del mismo modo, partiendo del caso más sencillo: una esfera {C, r} cortada por un plano {P_0, **n**}, perpendicular al dibujo. La intersección es un disco, centrado en el pie de la perpendicular desde el centro al plano: $C_\perp = C-(\mathbf{c}\cdot\mathbf{n})\mathbf{n}/n^2$ y su radio se obtiene por el teorema de Pitágoras: $r'^2 = r^2-(C_\perp-C)^2$. Si es menor que cero, no hay intersección; si es cero hay tangencia.

Para una recta {P_0, **t**}, que intercepta una esfera {C, r}; el pie de la perpendicular es $C_\perp = P_0+(\mathbf{c}\cdot\mathbf{t})\mathbf{t}/t^2$ y las intersecciones son los dos puntos $I_{0/1} = C_\perp\pm r'\mathbf{t}/|\mathbf{t}|$.

Si entendemos el dibujo de arriba como un cilindro infinito {C, **e**, r}; de eje {C, **e**}, perpendicular al dibujo; y que está interceptado por una recta inclinada {P_0, **t'**}; estamos viendo las componentes de los vectores en el plano del dibujo: $\mathbf{c}' = C-P_0$, $\mathbf{c} = \mathbf{c}'-(\mathbf{c}'\cdot\mathbf{e})\mathbf{e}/e^2$; $\mathbf{t} = \mathbf{t}'-(\mathbf{t}'\cdot\mathbf{e})\mathbf{e}/e^2$.



Los dos puntos $I_{0/1}$, calculados como antes, pero con estos **t** y **c**, están en la línea proyectada $P_0+\alpha\mathbf{t}$, con $\alpha^{0/1} = (I_{0/1}-P_0)\cdot\mathbf{t}/t^2$. Gracias a la invariancia afín, esos mismos α sirven en la línea inclinada, los puntos de intersección son: $I'_{0/1} = P_0+\alpha^{0/1}\mathbf{t}'$.

Si fuese un cilindro finito, los límites vendrían definidos por dos puntos del eje: $C+h_0\mathbf{e}$ y $C+h_1\mathbf{e}$. Cada intersección efectivamente estará en la superficie lateral del cilindro si $(I'_{0/1}-C)\cdot\mathbf{e}/|\mathbf{e}|$, está entre h_0 y h_1

En estos casos sí importan los puntos de intersección, que se utilizan para el mapeo de texturas.

Esto cubre los casos más usuales en CG; para *ray-tracing*, interacción o colisión y mapeo de texturas. Cualquier otro se puede lograr con los mismos principios.

Optimización

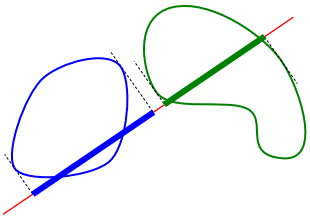
La implementación de algoritmos geométricos debe hacerse con mucho cuidado para obtener las prestaciones que se plantearon en la introducción. Primero se implementan las ecuaciones tal como se deducen o se encuentran, hasta lograr que el programa funcione robustamente (siempre entregue un resultado útil) y con la precisión adecuada. Lo más difícil es siempre evitar divisiones por cero y errores de precisión numérica. **Luego** se puede ajustar la eficiencia (optimización), sin perder los otros atributos, midiendo tiempos para miles o millones de intersecciones; evitando utilizar algoritmos cuyo costo computacional en tiempo y memoria sea mayor a O(n log(n)) en la cantidad de argumentos; tratando de minimizar los fallos de *cache* y utilizando técnicas paralelizables, cuando sea posible.

El mecanismo de optimización depende fuertemente del problema. Un ejemplo muy importante de intersecciones es el de rayo contra triángulo; la mayoría de los objetos vienen dados mediante *b-rep*, es decir: triangulaciones. En tal caso conviene averiguar, en primer lugar, si habrá intersección; es decir si los parámetros correspondientes al triángulo estarán entre cero y uno. El parámetro en el rayo se calcula solamente si se requiere, por ejemplo: la distancia al ojo (*z-buffer*) o, al menos, averiguar si la intersección está delante del ojo. Por otro lado, también es muy frecuente el cálculo de la intersección de un único plano con una gran cantidad de segmentos; por ejemplo, cuando se corta un objeto (sus aristas) mediante un plano para hacer *clipping*; en este caso, se calcula el parámetro (cero) en el segmento y solo cuando sepamos que habrá intersección; los parámetros en el plano no importan.

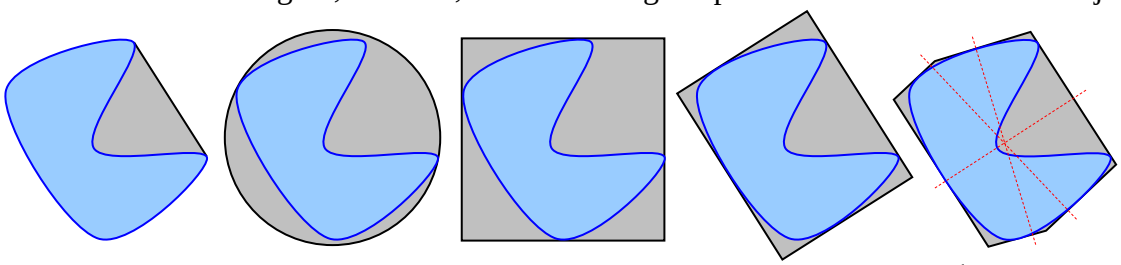
Las técnicas que veremos a continuación permiten el ordenamiento espacial y la simplificación de los objetos para responder rápidamente a la pregunta: "¿Puede haber intersección?"

Línea separadora y envoltorios

Dos objetos no se interceptan si sus proyecciones sobre alguna línea no se solapan; equivalentemente: si existe un plano que los separe, dejando un objeto a cada lado. Encontrar una línea separadora puede ser mucho más complicado e ineficiente que calcular la intersección; por lo tanto, se realiza un par de intentos con proyecciones sencillas; por ejemplo, las coordenadas (las líneas son los ejes cartesianos) o la línea que une los centros.



Para los objetos complicados se suele recurrir a envoltorios convexos simples (bolsas o cajas). Separar los envoltorios es más sencillo que separar los objetos. El análisis de intersecciones suele hacerse primero con el envoltorio elegido; si existe, se analiza luego la posible intersección con el objeto.



De izquierda a derecha: el *convex-hull* o mínimo envoltorio convexo; la esfera envolvente mínima o *bounding-sphere* o *enveloping-sphere*; el *axis-aligned bounding-box* o *bounding-box* a secas, que podría traducirse como "caja envolvente", de caras paralelas a los planos de coordenadas; luego el *oriented bounding-box* o caja orientada, definida una dirección conveniente y otra perpendicular, en general son los ejes de inercia del objeto (*object-oriented bounding-box*); finalmente, el *k-discrete oriented polytope* que tiene *k* pares de caras en direcciones dadas. Todos tienen extensión obvia a tres o más dimensiones.

El AABB se calcula mediante los máximos y mínimos de cada coordenada. Los alineados: OOBB y k−DOP, con el máximo y mínimo en un conjunto de direcciones $\{\mathbf{d}_i\}$ (min_max($\mathbf{x}\cdot\mathbf{d}_i$)). La mínima esfera, el menor OOBB y el *convex-hull* son difíciles de calcular y requieren herramientas de Geometría Computacional. Todos se pueden utilizar aproximados y predefinidos al crear los objetos. El CH es especialmente útil cuando viene dado por definición, por ejemplo, en las NURBS. La esfera aproximada provee la sencillez del cálculo de sus intersecciones con otras o con rayos; es el envoltorio más sencillo y utilizado en juegos; ejemplos: la colisión o intersección entre dos objetos se descarta si los centros están a mayor distancia que la suma de radios. La intersección de un objeto con un rayo se descarta si el rayo dista del centro más que el radio.

Ordenamiento espacial

El análisis de intersección, colisión o proximidad de objetos es esencialmente cuadrático: todos contra todos. Una rutina de orden cuadrático en el número de datos se puede hacer mucho menos costosa si se subdivide el espacio en cajas (*bins* o *buckets*) o celdas, que contienen unos pocos objetos y se analiza el problema caja por caja. El paradigma de *Divide and Conquer* se basa en que la $\sum$ pocos*pocos << todos*todos. El "mejor" método para repartir objetos en cajas depende del problema; se pueden preordenar los objetos, sus vértices o sus envoltorios, dependiendo del problema en particular. Si un objeto ocupa más de una celda, igual el tiempo se reduce a analizar un par o muy pocas celdas. El armado de las cajas se hace con estructuras y algoritmos estándar y cada tipo de subdivisión está muy estudiado, solo se discute cual utilizar en cada caso y ligeras variantes. Obviamente, hay que considerar también el tiempo de división del conjunto en cajas: el tiempo insumido en armar la estructura de datos.

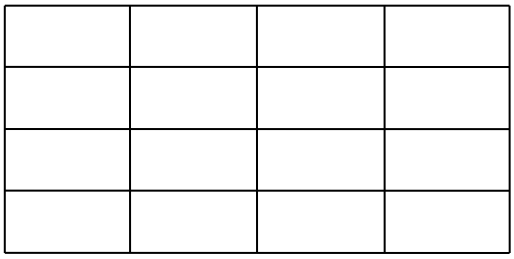
Las mismas técnicas que se aprendieron para el ordenamiento de números (1D) son generalizables para el ordenamiento espacial, esa es casi una definición de la Geometría Computacional (algoritmos y estructuras de datos multidimensionales). En particular, los árboles de divisiones recursivas reducen mucho el costo algorítmico, en general a O(n log(n)) y la búsqueda de un elemento cercano a O(1).

El ordenamiento más sencillo que puede pensarse es una simple subdivisión del espacio en cajas o celdas idénticas. Normalmente entendemos por "espacio" el AABB del conjunto de objetos (si son móviles será el máximo posible). Esta técnica consiste en dividir cada coordenada en un número adecuado de partes iguales. El principal problema aquí es la gran cantidad de cajas vacías; si la estructura queda muy desbalanceada (muy distinta cantidad de objetos por celda) la reducción de tiempo no es muy grande. Una variante de este método, que se utiliza en los juegos en primera persona con recorridos por salas separadas, consiste en la obvia agrupación de objetos en salas, más una relación de visibilidad a través de puertas y ventanas; se denominan métodos de ***cells and portals*** o celdas y portales.

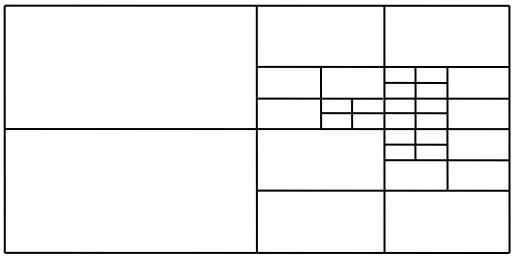
Un primer refinamiento de la división en partes iguales consiste en subdividir recursivamente las cajas por la mitad, cuatro partes iguales en 2D u ocho en 3D; pero solo cuando la caja contiene o intercepta más de determinada cantidad de objetos. El método se denomina ***quadtree*** en 2D y ***octree*** en 3D.

Se puede utilizar una división más simple, que además puede extenderse mucho mejor a varias dimensiones: el ***k-d tree***, o árbol *k*-dimensional es una partición recursiva con un plano a la vez. Cada plano puede partir a la mitad o por la mediana a la coordenada más larga de la celda o del BB de los datos de la celda o simplemente ciclando los planos cartesianos (*x, y, z, x, y*, …). Hay muchas variantes, de acuerdo al método de selección y ubicación del plano de corte. Si los planos no están alineados con los coordenados se denomina ***BSP-tree*** (*binary space partition tree*) y se utiliza con mucha profusión en el ambiente de los desarrolladores de juegos.

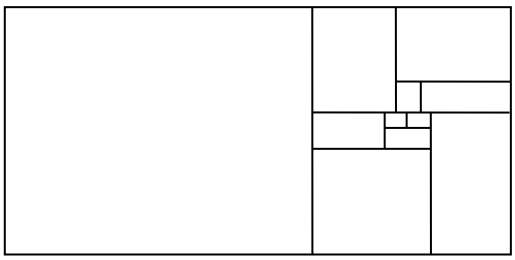
División regular



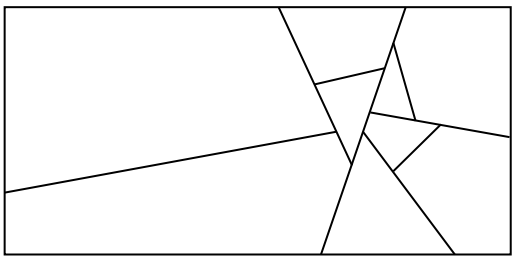
Quadtree



k-d Tree



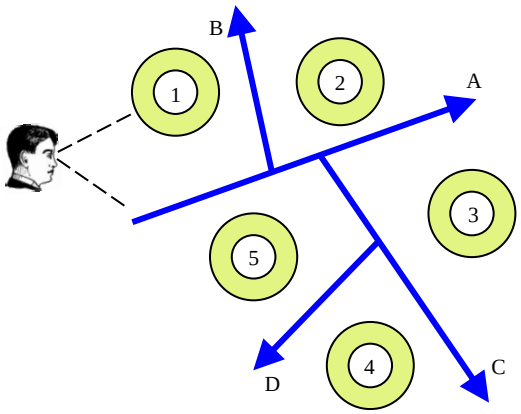
BSP Tree

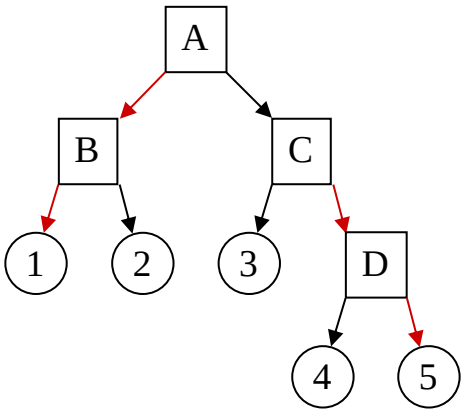


Para el *k-d-tree*, se suelen ordenar los datos por coordenadas y partir con la mediana del conjunto de cada celda. Esto produce **árboles balanceados**, es decir equilibrados en cantidad de elementos, para cada nivel de subdivisión, con ello se logran menos subdivisiones y menor profundidad total del árbol.

La selección del método y el balance costo/eficiencia dependen, obviamente, del problema particular.

Ejemplo: Algoritmo del Pintor mediante BSP-Tree.
Se trata de renderizar en forma ordenada, desde el fondo hacia el frente; por ejemplo, para objetos semitransparentes; aunque, en general, se utiliza para acelerar la oclusión, sin utilizar el *z-buffer*.





La explicación y el dibujo son bidimensionales, pero la técnica es exactamente igual en el espacio; cada divisor será un plano en lugar de una recta. Normalmente no se buscan los divisores óptimos que no recorten objetos; en los juegos, esta construcción se hace en forma manual, para los objetos fijos; se hace una sola vez y se reutiliza durante todo el juego para el renderizado rápido.

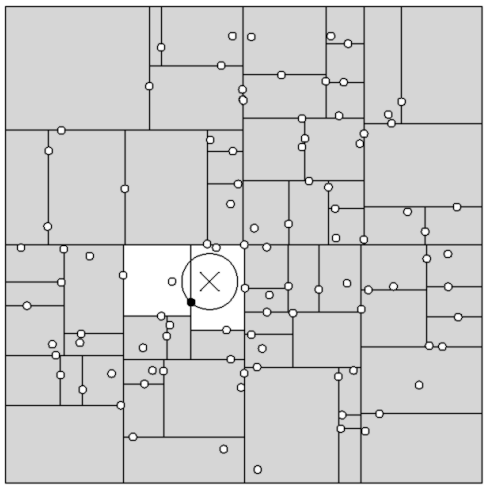
La primera celda es el espacio completo. A medida que se fija un divisor, la celda se divide en dos y sus objetos se asignan a cada lado del divisor, construyendo un árbol de relaciones izquierda-derecha, según la dirección de la semirrecta; o arriba-abajo, en 3D, según la normal del semiplano.

Con el árbol ya construido (probablemente permanente) se indica, para cada divisor, de qué lado está el observador móvil. En la figura se indicó en rojo la partición que contiene al observador en cada división.

El orden de renderizado se obtiene recorriendo el árbol en forma ordenada, primero por donde no se encuentra el observador. La secuencia de recorrido es: A C 3 C D 4 D 5 D C A B 2 B 1 B A, dando un orden de renderizado: 3, 4, 5, 2, 1. Notar que el objeto 5 está más cerca del ojo que el 2, pero aun así nunca podría ocultarlo; es decir que es correcto recorrer y renderizar primero, y en forma completa, las ramas que no contienen al observador.

Ejemplo: Búsqueda del punto más cercano con un *k-d Tree.*

El árbol de la figura se construyó utilizando los puntos como divisores y admitiendo un solo punto interior en cada hoja; pero puede hacerse mucho menos profundo si se permite que queden unos pocos puntos en cada hoja. Lo mismo para un *quadtree* u *octree*.



El árbol se construye con algún algoritmo que se encuentre disponible, las estructuras y métodos se pueden entender fácilmente, pero resultaría muy extenso explicarlas aquí. Los nodos del árbol son celdas con un divisor; las hojas son las celdas finales con uno o pocos puntos interiores y algunos en los bordes.

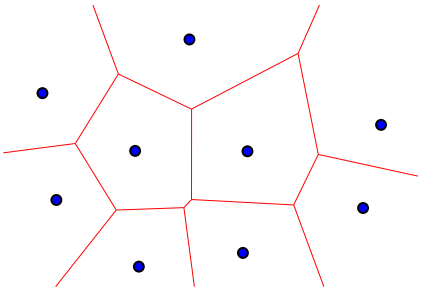
Para encontrar el punto más cercano a un punto dado (marcado con una x en el dibujo), primero se busca la celda que lo contiene. Se recorre el árbol desde el nodo raíz (el BB de los datos) y, en cada nodo, se identifica de qué lado del divisor queda el punto y se pasa a ese hijo, hasta llegar a la celda-hoja que lo contiene.

Entre los puntos de la celda, se busca el más cercano; la menor distancia define un radio inicial para la búsqueda posterior, pues puede haber puntos más cercanos en celdas vecinas. Ese radio define una circunferencia o, más sencillamente, un cuadrado, que es su BB.

Nuevamente se busca en el árbol; pero ahora, en todas las hojas que intercepten o contengan a la circunferencia o su BB. Cada vez que se encuentra un punto más cercano que el actual, se reduce el radio de búsqueda.

Diagrama de Voronoï y triangulación Delaunay

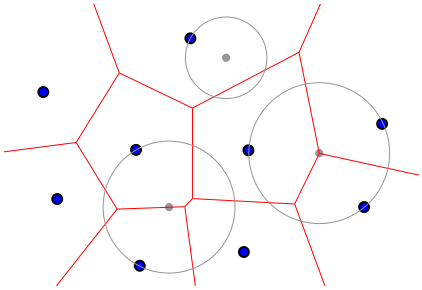
El diagrama de Voronoï es la última técnica de ordenamiento espacial que trataremos; pero le daremos una atención especial, debida a sus propiedades y múltiples aplicaciones.



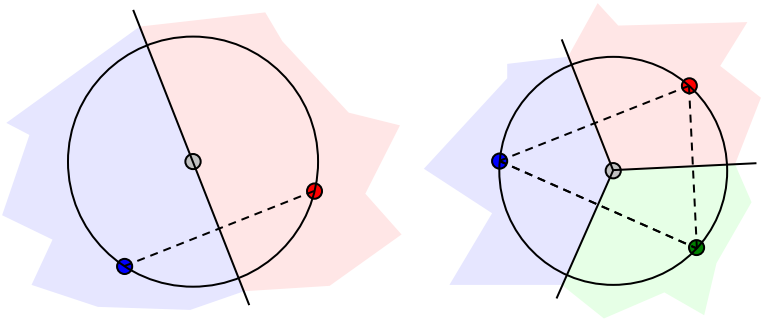
En forma general, consiste en dividir el espacio en celdas, una para cada objeto; de modo que cada celda contiene los puntos del espacio que están más cerca de su objeto que de cualquier otro. Las múltiples variantes se deben al tipo de objetos y a la forma de medir la distancia. Aquí estudiaremos sólo el diagrama de Voronoï de puntos aislados y la distancia euclídea. El análisis lo haremos en 2D, de un modo que permite analizar e implementar el mismo procedimiento en 3D.

A los puntos dato (fijos) los llamaremos nodos, para diferenciarlos de los puntos comunes del espacio.

En la figura de arriba se muestra un conjunto de nodos y su diagrama de Voronoï. **Todo** el espacio se divide en celdas convexas, cada una es el conjunto de puntos más cercanos a su nodo que a otro.



Para entender la geometría del problema, tomamos cualquier punto del plano y hacemos crecer una circunferencia centrada en el mismo. Si el punto estaba en el interior de alguna celda, la circunferencia encuentra primero al correspondiente nodo, el más cercano al punto. Si estaba en una arista, encuentra dos nodos a la vez, la arista separa las dos celdas de los dos nodos equidistantes del punto. Finalmente, si el punto es un vértice del diagrama, donde se encuentran tres (o más) celdas, equidista de tres (o más) nodos a la vez.

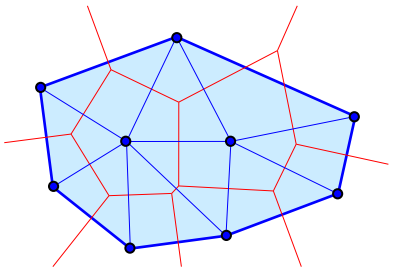


Las aristas del diagrama de Voronoï separan dos celdas y equidistan de sus dos nodos. Forman parte de la **mediatriz** de los dos nodos, que es la recta que parte por la mitad y es perpendicular al segmento que los une. Los vértices, donde se juntan tres mediatrices, son a su vez el centro de la circunferencia que circunscribe al triángulo formado por los tres nodos. En 3D sucede lo mismo, las mediatrices son planos y las circunferencias esferas.

El concepto y propiedad más importante es que un vértice del diagrama equidista de tres nodos y por lo tanto es el centro de la circunferencia que circunscribe al triángulo que forman. Lo mismo sucede en 3D con esferas, cuatro nodos y el tetraedro que forman.

Los triángulos asociados a los vértices del diagrama forman una triangulación. Una “triangulación de un conjunto de nodos” es una partición de su *convex-hull* (CH) en triángulos, de modo que:

1) Sólo los nodos y todos los nodos son vértices de los triángulos.
2) Dos triángulos comparten una arista completa o un nodo o nada
   - ⇒ No hay uniones en T.
   - ⇒ Ningún triángulo tiene nodos en el interior.
   - ⇒ Cada arista interior es compartida por dos triángulos;
   - ⇒ No hay triángulos solapados.



Las aristas del CH son las únicas que pertenecen a un solo triángulo; sus nodos son los únicos que pertenecen a celdas de extensión infinita. Esas características se pueden utilizar para determinar el CH del conjunto.

Dado el conjunto de nodos, hay muchas triangulaciones posibles. Esta triangulación, que se obtiene como “dual” del diagrama de Voronoï, se llama Triangulación Delaunay (se pronuncia: Deloné).

Esta dualidad es un concepto extendido de la teoría de grafos. Cada vértice del diagrama es centro de la circunferencia que equidista de los nodos y define el triángulo y se puede ver además la relación uno a uno entre aristas (segmento - mediatriz) y entre celdas y nodos.

La siguiente tabla especifica en dos y tres dimensiones la dualidad de elementos.

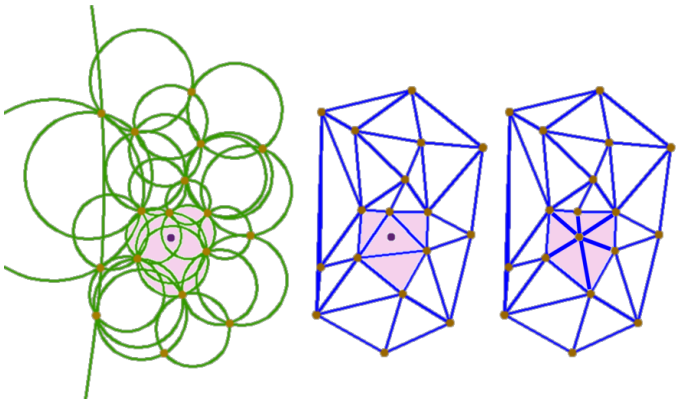
2D	
Celda Voronoï	**Triángulo Delaunay**
Celda	Nodo
Arista	Arista
Vértice	Triángulo

3D	
Celda Voronoï	**Tetraedro Delaunay**
Celda	Nodo
Arista	Cara
Cara	Arista
Vértice	Tetraedro

Hay varios métodos propuestos y eficientes para construir un diagrama de Voronoï, o su equivalente dual: la triangulación Delaunay. El método conceptualmente más sencillo aprovecha la característica distintiva de la triangulación Delaunay: las circunferencias no contienen nodos en el interior.

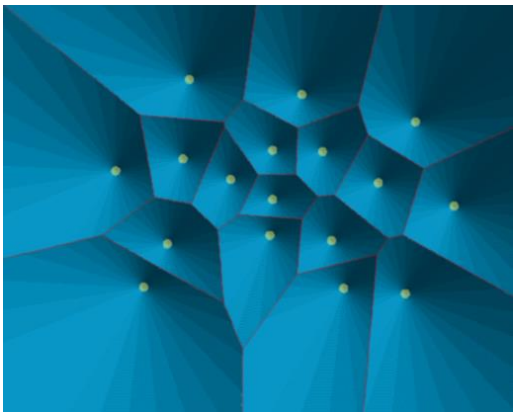
La construcción incremental y aleatoria se realiza agregando los nodos, de a uno y elegido al azar, en una triangulación preexistente. Empieza con un gran triángulo, de nodos virtuales (inexistentes) que envuelve holgadamente a todo el conjunto. Al agregar un nodo, se buscan las circunferencias que lo contienen y, por lo tanto, deben desaparecer. La reconstrucción es local, se limita a la unión de las circunferencias que contienen al nuevo nodo, es el entorno natural (***natural neighborhood***) del nodo.

El conjunto de triángulos, cuyas circunferencias contienen al nuevo nodo, forma un polígono que envuelve al nodo y se denomina “cavidad del nodo”. La cavidad puede no ser convexa, pero toda su frontera es visible desde el nodo. Esta se reconstruye con nuevos triángulos; cada uno formado por el nodo y una arista exterior de la cavidad. Para cada nuevo triángulo se calcula su circunferencia. En 3D la cavidad es un poliedro y el procedimiento es igual. Al finalizar se eliminan los nodos virtuales y sus triángulos adyacentes.



Cuando existen más de tres puntos en la misma circunferencia, el diagrama de Voronoï tiene un vértice equidistante de más de tres puntos. En tal caso, la triangulación genera problemas numéricos que se suelen salvar perturbando la posición del nodo entrante, para restaurar la condición normal. En 3D el problema es mucho mayor, porque la cocircularidad genera tetraedros aplastados conocidos como *slivers*; hay una inmensa cantidad de técnicas publicadas para eliminarlos, pero es muy complicado.

El diagrama de Voronoï se obtiene naturalmente en procesos de crecimiento, que parten desde “semillas” fijas. Puede verse, por ejemplo, en el caparazón de una tortuga, la piel de una jirafa, las alas de un insecto, en el crecimiento de granos apiñados o en el crecimiento de cristales en las aleaciones.



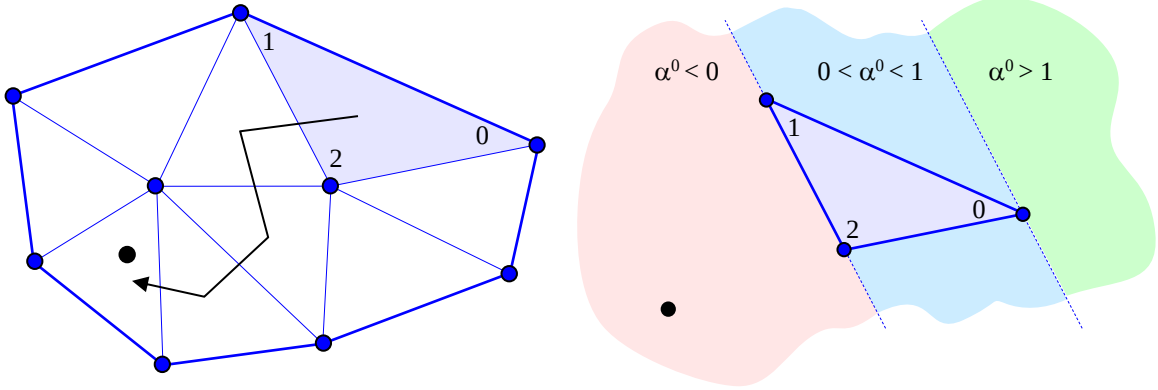
En la figura se muestra construido como una intersección de conos vista desde arriba. Los conos simulan el crecimiento a velocidad constante desde los nodos (crece el diámetro a medida que aumenta la profundidad). Como método constructivo es extremadamente ineficiente, pero permite visualizar el diagrama con unas pocas líneas en OpenGL.

El diagrama de Voronoï y la triangulación Delaunay tienen propiedades muy fructíferas en computación gráfica y geométrica, debido a que codifican las relaciones de proximidad y vecindad que, de otro modo, serían muy costosas de construir. Sobran ejemplos. Podemos averiguar en forma muy eficiente cuales son los nodos cercanos a un determinado punto, averiguando a que triángulo pertenece. Podemos planear el movimiento automático de un robot, esquivando objetos de la escena, si lo obligamos a moverse por las líneas del diagrama de Voronoï. ¿Dónde conviene poner una nueva farmacia? En el diagrama de Voronoï de las farmacias existentes, elegimos el centro de la circunferencia cuyo radio r sea el más grande; es un vértice del diagrama que no tiene ninguna farmacia a menos de r “a la redonda”. Además, la triangulación Delaunay es el método de elección obvia para interpolar datos de puntos desorganizados.

Búsqueda lineal o *linear walk*

Para muchas operaciones de ubicación de puntos o búsqueda de proximidad y en particular para la construcción incremental del diagrama de Voronoï, se requiere saber a qué esferas o circunferencias pertenece un punto (el nuevo nodo a insertar) y podemos averiguarlo respondiendo a qué tetraedro o triángulo pertenece. Un proceso muy eficiente para hacerlo se denomina *linear walk.*

Tenemos una triangulación (no necesariamente Delaunay, pero si convexa) de un conjunto de nodos y queremos consultar a qué triángulo pertenece un punto del plano. El procedimiento consiste en partir de un triángulo cualquiera y movernos por vecindades, de un modo recursivo que nos acerque cada vez más al punto. Para ello debemos contar con una estructura de datos que identifique, para cada triángulo, cuáles son sus vecinos (por arista en 2D; por cara de tetraedros en 3D), ordenamos la lista de modo que el vecino i-ésimo sea opuesto al i-ésimo nodo del triángulo ($i \in \{0,1,2\}$). En los triángulos vecinos del exterior, podemos asignar el puntero nulo o un *flag* (-1) al exterior del CH en las aristas de frontera.



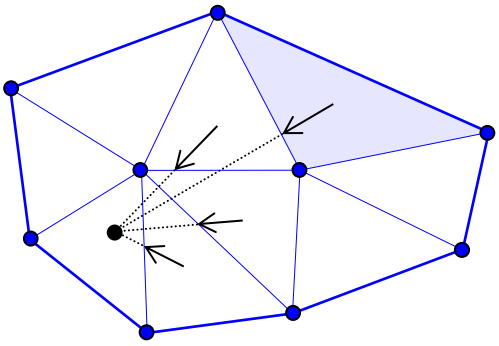
Siguiendo la figura anterior, partimos del triángulo pintado y calculamos las coordenadas baricéntricas en el punto (sin dividir por el área total). La línea del segmento opuesto a cada nodo separa el semiplano positivo del negativo. La coordenada baricéntrica más negativa nos indica, entonces, a que vecino pasar. Cuando son todas positivas, es que se encontró el triángulo que lo contiene.

El nodo más cercano al punto puede no ser un vértice del triángulo, pero si debe estar entre los que definen las circunferencias que contienen al punto. Las circunferencias se hallan localmente, buscando entre los vecinos de cada una hallada, empezando por el triángulo que contiene al punto.

Cuando hay muchísimos triángulos, el proceso de búsqueda lineal se acelera si en lugar de partir de cualquier triángulo se parte de uno cercano. Se precalcula una estructura de ordenamiento espacial para el conjunto de nodos, puede ser un *quadtree* con unos pocos nodos por hoja, basta incorporar el punto buscado en la estructura y partir desde algún triángulo de un nodo de la misma celda terminal u hoja.

Cuando el punto se mueve, conviene partir del triángulo en el que estaba antes. Del mismo modo, cuando se trata de ubicar una sucesión de puntos contiguos (por ejemplo, píxeles), conviene comenzar por el que contenía al anterior.

Cuando se posee cualquier tipo de polígonos o poliedros, no necesariamente simpliciales (triángulos o tetraedros) el proceso de búsqueda se puede realizar también buscando una cara o lado que intercepte el rayo que une un punto interior del polígono con el punto buscado. El ejemplo más común es ubicar en que sala está el personaje móvil en un juego, o que sala señala el jugador con el cursor o el arma. También se utiliza para mover el personaje de un lugar a otro evitando obstáculos.

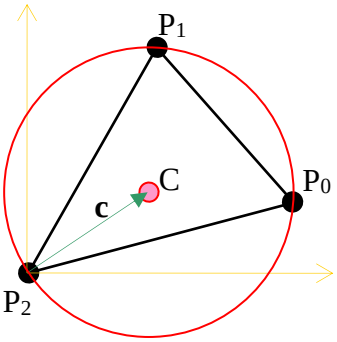


Apéndice: Circunferencia de tres puntos y esfera de cuatro puntos.

En el proceso de armado del diagrama de Voronoï, se crean y destruyen esferas o circunferencias con cada punto incorporado; por lo tanto, el algoritmo utilizado para definirlas debe implementarse con la mayor eficiencia y robustez posibles. El análisis siguiente es válido en cualquier cantidad de dimensiones, pero fijemos ideas en 2D.

Queremos el centro C y radio r de la circunferencia definida por los tres puntos $\{P_0, P_1, P_2\}$. La ecuación que la define plantea que el radio es la distancia desde cada punto al centro:

$$r^2 = (C-P_i)^2 = C^2 - 2\,C\cdot P_i + P_i^2$$



En 2D son tres ecuaciones ($i \in \{0,1,2\}$) y tres incógnitas (C_x, C_y, r^2), pero cuadráticas. Restando P_2 a cada punto: $\mathbf{c} = C-P_2$; $\mathbf{p}_i = P_i-P_2$. La tercera de las ecuaciones anteriores queda:

$$r^2 = (C-P_2)^2 = c^2,$$

reemplazando r^2 en las dos primeras ecuaciones, queda:

$$c^2 = c^2 - 2\,\mathbf{c}\cdot\mathbf{p}_i + p_i^2 \quad \Rightarrow \quad \boxed{2\,\mathbf{c}\cdot\mathbf{p}_i = p_i^2} \quad (i \in \{0,1\})$$

De ese sistema lineal 2x2 (o 3x3 en 3D) se calculan $\mathbf{c}$ y su módulo r, luego se ubica $C = P_2 + \mathbf{c}$.

Bien mirado, el sistema dice que el centro se proyecta en el medio de cada arista: $\mathbf{c}\cdot\mathbf{p}_i/|\mathbf{p}_i| = |\mathbf{p}_i|/2$, cosa que ya sabíamos porque está definido por la intersección de las tres mediatrices.

El determinante de los coeficientes es un múltiplo del área del triángulo (volumen del tetraedro, etc.).

Si se prefieren fórmulas cerradas se pueden utilizar:

$2a^2\,\mathbf{c} = \mathbf{a}\times(p_1^2\,\mathbf{p}_0 - p_0^2\,\mathbf{p}_1)$ (Circunferencia, se restó P_2; $\mathbf{a} = \mathbf{p}_0\times\mathbf{p}_1$)

$2v\,\mathbf{c} = \Sigma\, p_i^2\,\mathbf{p}_i^*$ (Esfera: se restó P_3; $\mathbf{p}_i^* = \mathbf{p}_{(i+1)\%3} \times \mathbf{p}_{(i+2)\%3}$; $v = \mathbf{p}_0\times\mathbf{p}_1\cdot\mathbf{p}_2$)