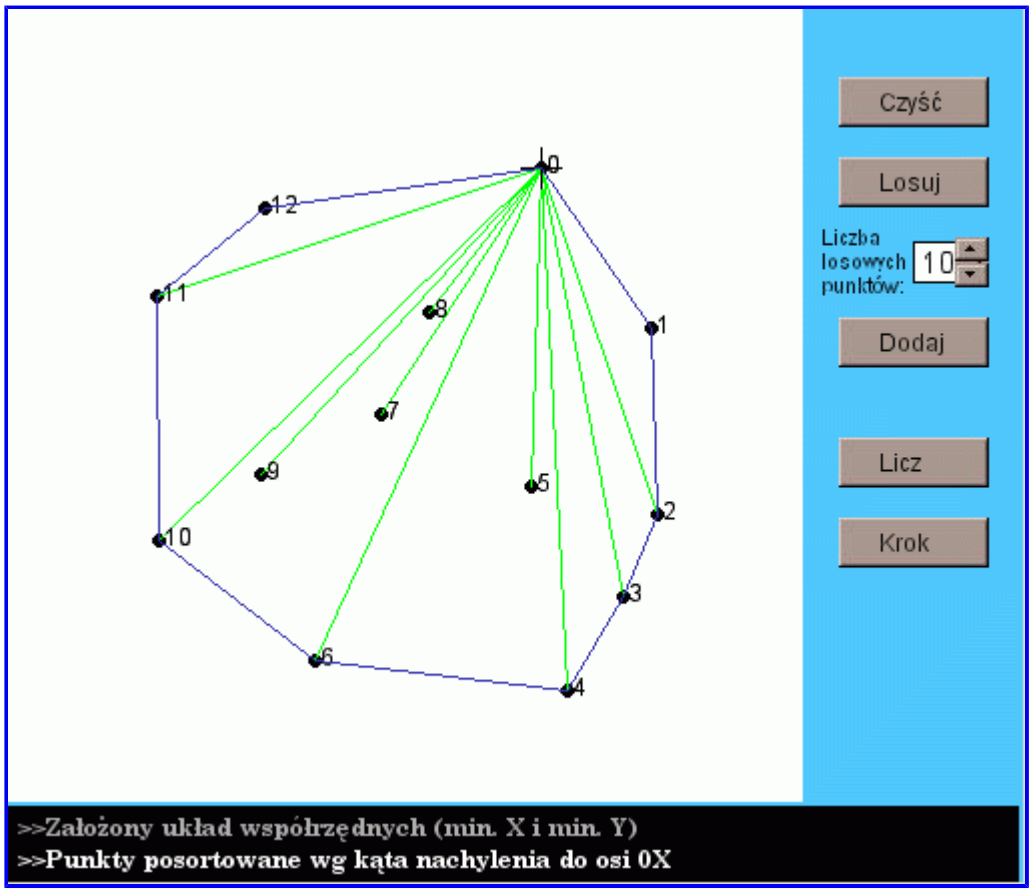
Lekcja 8: Algorytmy geometryczne

Wstęp

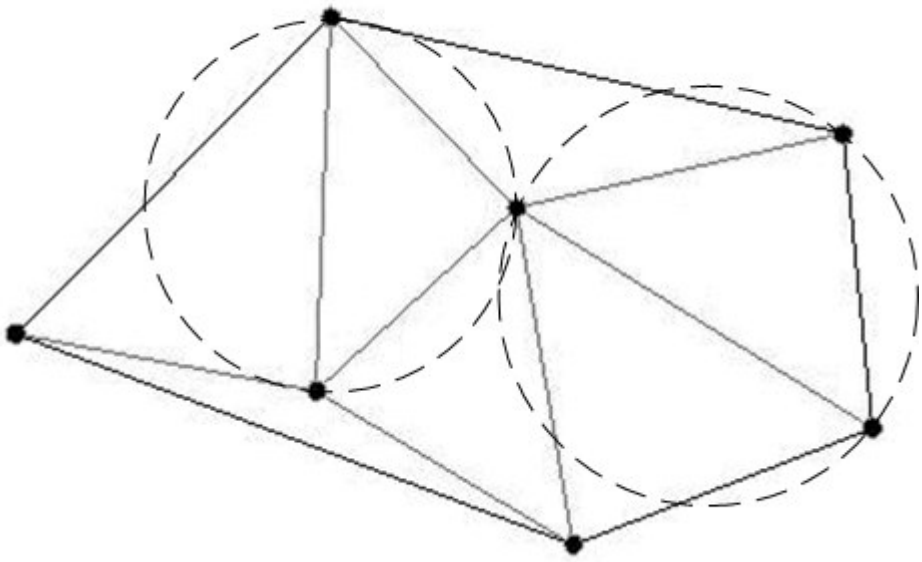
Na zakończenie tego kursu proponujemy Wam krótką lekcje przeglądową na temat najbardziej popularnych algorytmów geometrycznych. Tym razem nie będzie szczegółowego, precyzyjnego opisu algorytmów, a raczej krótki rzut oka na problemy, jakie w tej dziedzinie występują, z wykorzystaniem przygotowanych przez naszych studentów apletów i odwołaniem się do ciekawych zasobów sieciowych. Zwrócimy uwagę na znane już Wam kategorie algorytmów i struktury danych szczególnie przydatne w geometrycznych implementacjach.

Otoczka wypukła i triangulacja Delaunaya

Poszukiwanie otoczki wypukłej zbioru punktów stanowi jeden z najbardziej popularnych algorytmów geometrii obliczeniowej. Przez otoczkę wypukłą zbioru punktów na płaszczyźnie rozumie się najmniejszy wielokąt wypukły taki,że wszystkie punkty zbioru leżą albo wewnątrz wielokąta, albo na jego brzegu. Zadanie to bywa żartobliwie nazywane "ogradzaniem śpiących tygrysów". Znanych jest kilka różnych metod, które potrafią poradzić sobie z tym zadaniem w czasie rzędu O(n lg n). Jedną z najbardziej znanych jest **algorytm Grahama**, używający stosu, na którym przejściowo składane są punkty będące kandydatami na wierzchołki otoczki, i w razie potrzeby są one ze stosu zdejmowane. Wszystkie punkty są sortowane ze względu na współrzędną kątową ich położenia (liczonego w kierunku przeciwnym do ruchu wskazówek zegara) względem punktu o minimalnej współrzędnej y.Umiejętność implementacji algorytmu związana jest z koniecznością uwzględniania rozmaitych przypadków szczególnych, których w geometrii obliczeniowej jest dużo. Przykład implementacji możecie obejrzeć na załączonym aplecie:



Triangulacja Delaunaya jest to taki podział przestrzeni wyznaczonej zbiorem punktów na płaszczyźnie, który dzieli otoczkę wypukłą zbioru punktów na trójkąty w taki sposób, że żaden punkt nie leży wewnątrz okręgu opisanego na dowolnym trójkącie:

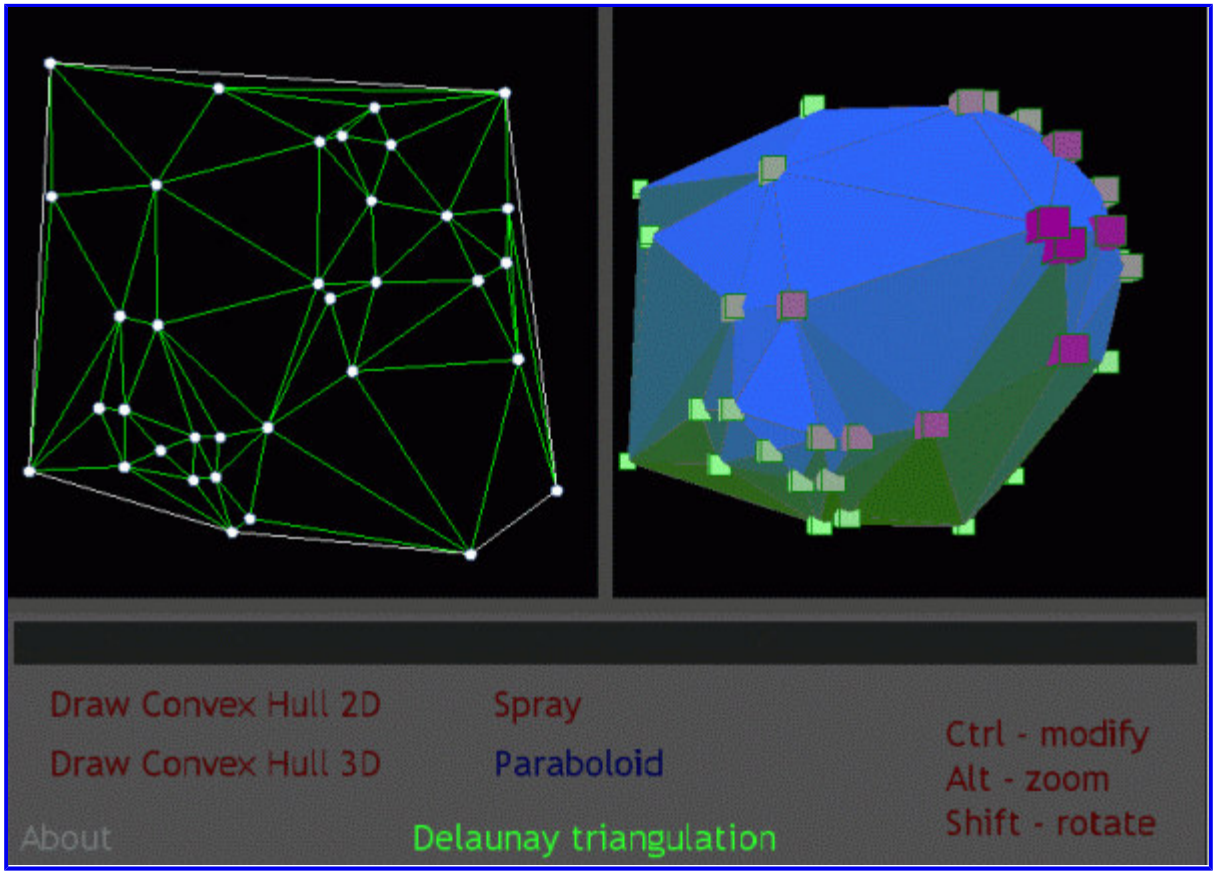


Tak uzyskany podział na trójkąty ma tę własność, że maksymalizuje minimalny kąt spośród wszystkich trójkątów należących do triangulacji. Dzięki temu w maksymalnie możliwym stopniu unika się trójkątów o małych kątach, czyli "długich i wąskich".

Triangulacja Delunaya znajduje zastosowanie w wielu różnych dziedzinach wymagających zastąpienia zbioru punktów siatką trójkątów. Najprostszy algorytm triangulacji jest algorytmem **przyrostowym**, polegającym na dodawaniu kolejnych punktów do zbioru trójkątów, sprawdzaniu otoczenia dodawanego punktu i retriangulacji tego otoczenia w razie potrzeby. Algorytm ma złożoność obliczeniową $O(n^2)$ i może być przyspieszony z wykorzystaniem algorytmu **zamiatania płaszczyzny**, pokazanego w rozdziale następnym.

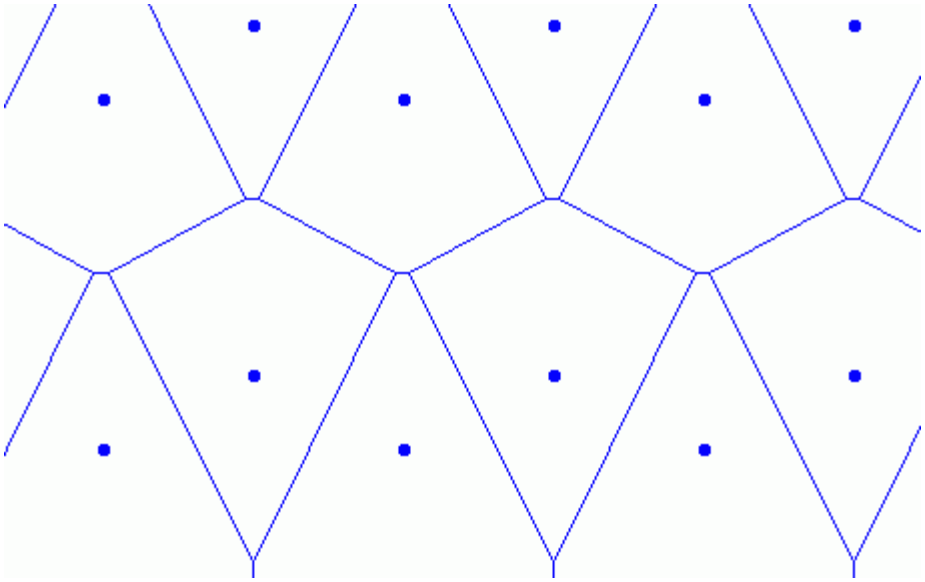
Inną metodą, która pozwala na obniżenie złożoności do poziomu O(n lg n), jest metoda **"dziel i zwyciężaj"**, która rekurencyjnie dzieli zbiór punktów na dwa zbiory, dla każdego wyznacza triangulację Delaunaya, a następnie łączy obie triangulacje w jedną wynikową. Jak zwykle, etap łączenia jest to najtrudniejszy krok tego algorytmu, szczególnie trudny w aspekcie obliczeń geometrycznych.

Kolejny bardzo ciekawy pomysł wyznaczenia triangulacji Delaunaya polega na wykorzystaniu faktu, że jest ona rzutem otoczki wypukłej 3D wyznaczonej dla punktów o współrzędnych (x, y, x^2+y^2), gdzie punkty (x,y) tworzą zbiór zadany. Ten właśnie algorytm został zaimplementowany w oryginalny sposób w aplecie, który prezentujemy poniżej; możecie tu w bardzo wygodny sposób dorzucać punkty do zbioru i obserwować otoczkę 3D zbudowaną na paraboloidzie (bardzo plastycznie zwizualizowaną)- oraz efekt końcowy, czyli wyznaczaną w trybie on-line triangulację Delaunaya.



Bardzo ciekawe aplety, które porównują czasy działania i stopień złożoności różnych algorytmów wyznaczających triangulację Delaunaya, możecie znaleźć **tutaj** oraz **tutaj**

Z triangulacją Delaunaya nierozerwalnie związany jest (jako tzw. jej graf dualny) **diagram Voronoi**, czyli taki podział przestrzeni na obszary wokół zadanych punktów zbioru, że wszystkie punkty obszaru leą bliżej zadanego punktu zbioru niż względem jakiegokolwiek innego punktu tego zbioru. Przykładem zastosowania jest podział terenu na obszary leżące w jak najbliższej odległości wokół masztów nadajników sieci komórkowych. Przykład diagramu wygenerowanego jednym z powyższych apletów przedstawia rysunek:



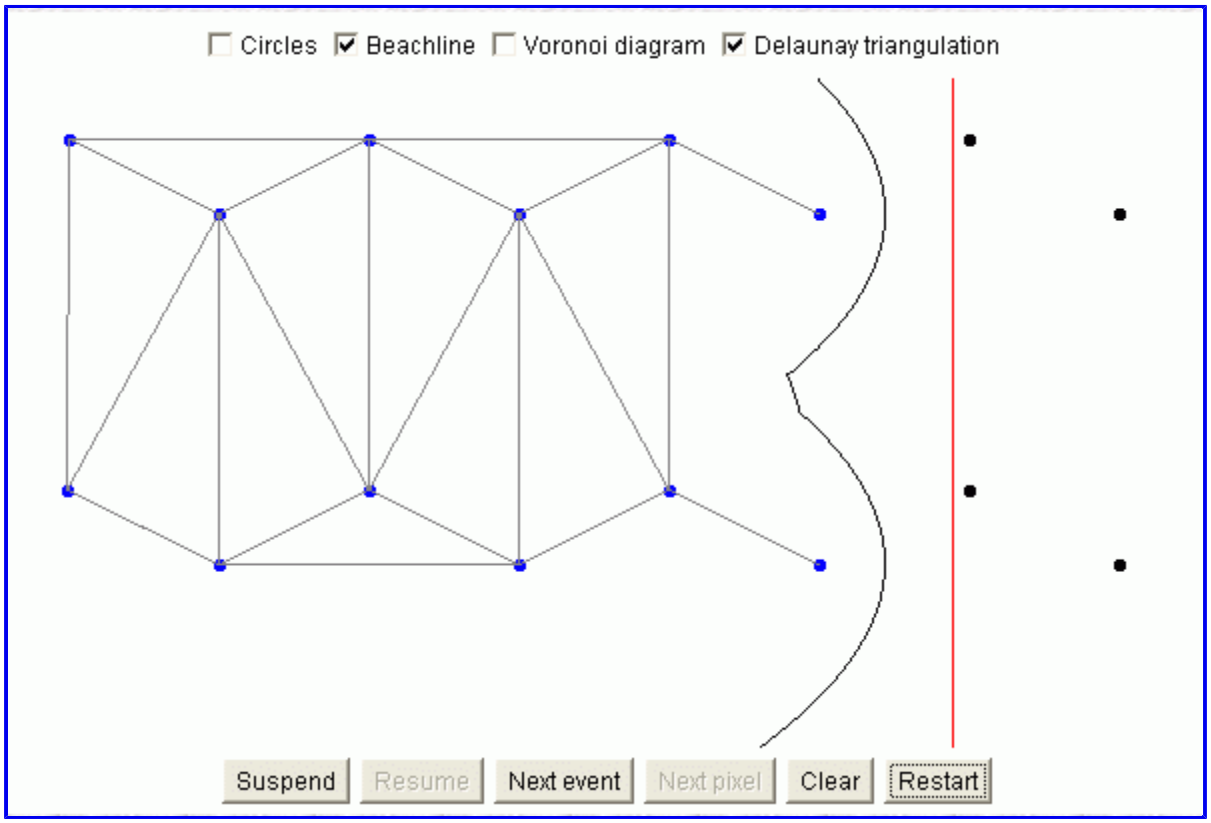
Metoda zamiatania płaszczyzny

Metoda zamiatania płaszczyzny (ang. *sweeping plane*) jest często spotykana w algorytmach geometrii obliczeniowej. Zastosowanie jej pozwala zmniejszyć złożoność obliczeniową wielu algorytmów z poziomu $O(n^2)$ do poziomu $O(n \lg n)$. Typowym przykładem zastosowania jest zagadnienie wyznaczania par przecinających się odcinków.

Pionowa prosta przesuwająca się na płaszczyźnie od strony lewej do prawej pełni rolę miotły, która pozwala systematycznie przeglądać pojawiające się podczas przesuwania obiekty geometryczne i sprawdzać zachodzące między nimi zależności. Zatrzymania miotły następują tylko w w tych charakterystycznych miejscach określonych współrzędną x i pełnią rolę **zdarzeń**.

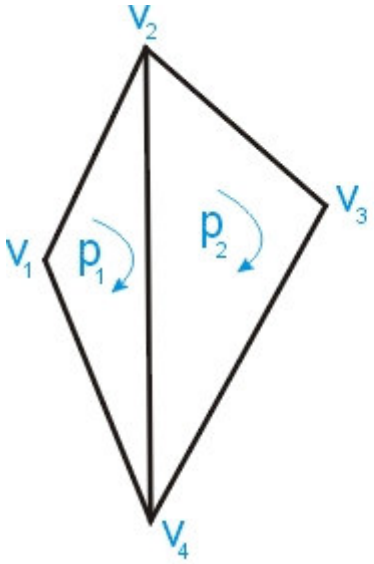
W przypadku algorytmu sprawdzania przecinania się odcinków zaczyna się od uporządkowania zbioru końców odcinków względem współrzędnej x (trzeba przy tym rozstrzygnąć problem, który się pojawia, gdy dwa końce mają taką samą współrzędną). Odcinki przechowywane są w strukturze **drzewa czerwono-czarnego**, która pozwala na efektywne (w czasie O(lg n)) operacje dodawania odcinków do zbioru, gdy miotła napotyka jego lewy koniec, i usuwania go ze struktury, gdy napotyka prawy koniec. Sprawdzenie przecinania się dwóch odcinków następuje wtedy, gdy po raz pierwszy stają się one sąsiadami w liniowo uporządkowanym (poprzez strukturę drzewa czerwono-czarnego) zbiorze odcinków.

Okazuje się, że zagadnienie wyznaczenia triangulacji Delaunaya i diagramu Voronoi również może być rozwiązane metodą zamiatania płaszczyzny. Znakomity aplet, który obrazowo pokazuje ideę tej metody, możecie uaktywnic klikając w poniższy obrazek, bedacy zrzutem ekranu z tego apletu, dostępnego na stronie **http://www.diku.dk/hjemmesider/studerende/duff/Fortune.**



Struktura half-edge w reprezentacji brył

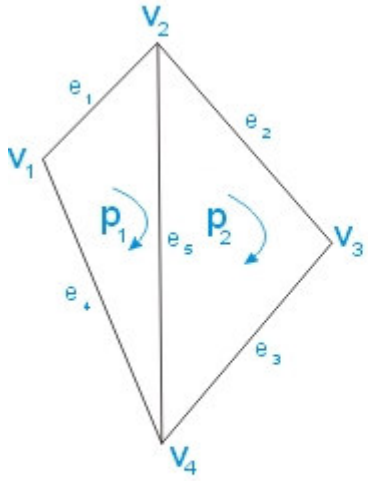
Na zakończenie tej lekcji pokażemy Wam przykład złożonej struktury listowej budowanej na wskaźnikach, która ma duże praktyczne znaczenie w reprezentacji brył. Chodzi o taką reprezentację, w której bryła jest aproksymowana ciągiem płaskich ścian. W wielu przypadkach taką strukturę zapamiętuje się bardzo prosto - jako listę wszystkich wierzchołków i listę ścian, z których każda określona jest przez adresy wierzchołków:



$$\mathbf{v} = \begin{cases} \mathbf{v_1} = (x, y, z) \\ \mathbf{v_2} = \ldots \\ \ldots \\ \mathbf{v_4} = \ldots \end{cases}$$

$$\mathbf{p} = \begin{cases} \mathbf{p_1} = (1,2,4) \\ \mathbf{p_2} = (2,3,4) \end{cases}$$

Bardziej złożone struktury polegają na tworzeniu dodatkowo listy krawędzi, dzięki czemu krawędzie przy wyświetlaniu nie są rysowane dwukrotnie (dla każdej przyległej ściany osobno); lambda na rysunku oznacza wskaźnik NULL/**nil**.

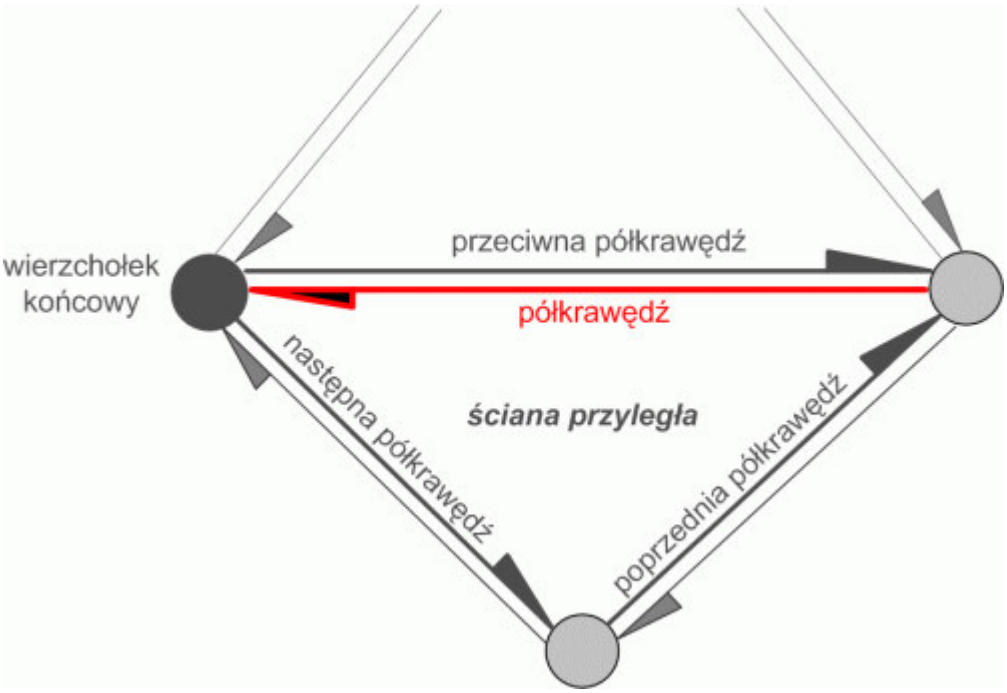


$$\mathbf{v} = \begin{cases} \mathbf{v_1} = \dots \\ \dots \\ \mathbf{v_4} = \dots \end{cases}$$

$$\mathbf{p} = \begin{cases} \mathbf{p_1} = (\mathbf{e_1}, \mathbf{e_5}, \mathbf{e_4}) \\ \mathbf{p_2} = (\mathbf{e_2}, \mathbf{e_3}, \mathbf{e_5}) \end{cases}$$

$$\mathbf{E} = \begin{cases} \mathbf{e_1} = (\mathbf{v_1}, \mathbf{v_2}, \mathbf{p_1}, \lambda) \\ \mathbf{e_2} = (\mathbf{v_2}, \mathbf{v_3}, \mathbf{p_2}, \lambda) \\ \mathbf{e_3} = (\mathbf{v_3}, \mathbf{v_4}, \mathbf{p_2}, \lambda) \\ \mathbf{e_4} = (\mathbf{v_4}, \mathbf{v_1}, \mathbf{p_1}, \lambda) \\ \mathbf{e_5} = (\mathbf{v_2}, \mathbf{v_4}, \mathbf{p_1}, \mathbf{p_2}) \end{cases}$$

Tu jednak chcemy pokazać strukturę jeszcze bardziej złożoną. Popatrzcie na rysunek:



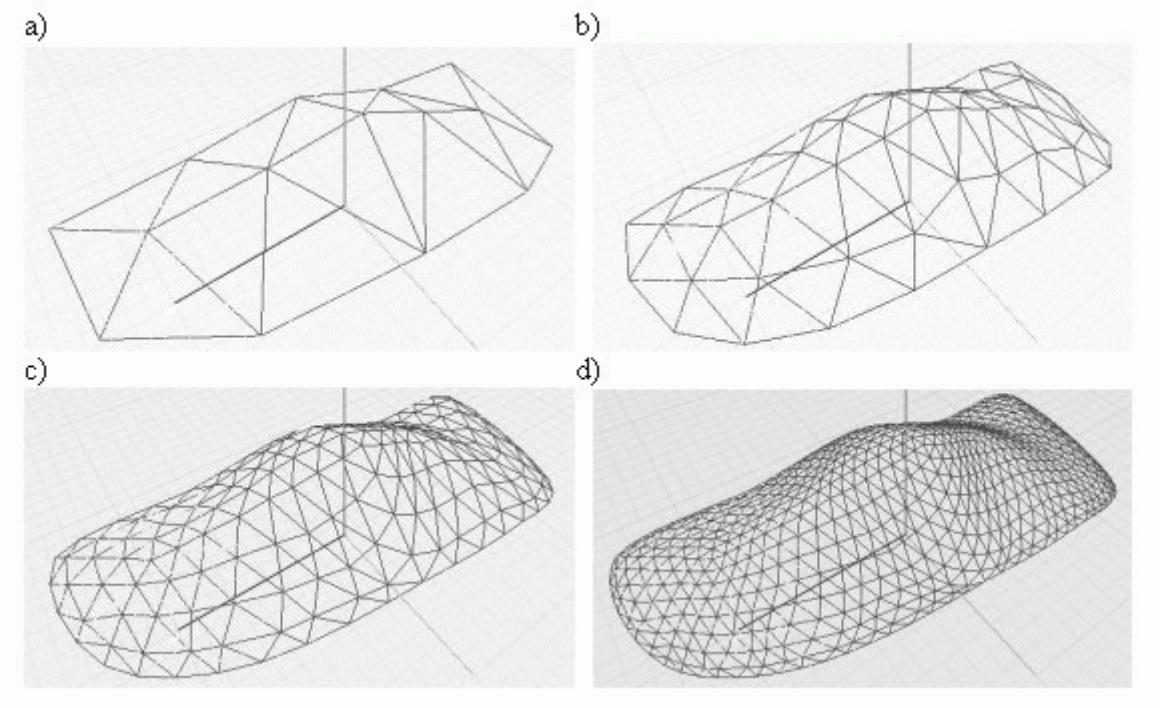
W tym przypadku każda krawędź jest reprezentowana przez dwie bliźniacze półkrawędzie, zwrócone w kierunkach przeciwnych. Każda półkrawędź (half-edge), zaznaczona tu na czerwono, zawiera wskaźnik na:

- półkrawędź przeciwną do danej
- wierzchołek końcowy półkrawędzi danej
- ścianę przyległą do danej półkrawędzi
- następną półkrawędź tej ściany, liczoną w kierunku przeciwnym do ruchu wskazówek zegara
- poprzednią półkrawędź tej ściany.

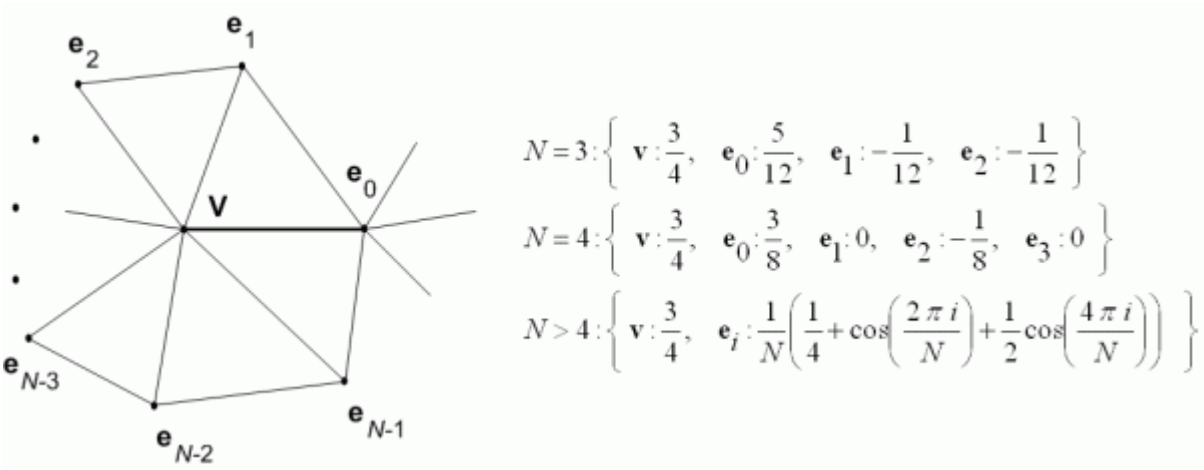
Jest to typowa, aczkolwiek nie minimalna reprezentacja struktury powszechnie zwanej **half-edge**. Minimalna postać half-edge zawiera tylko wskaźnik na półkrawędź przeciwną i następną, wygodniejsza w użyciu jest jednak struktura taka jak pokazaliśmy,

bardziej rozbudowana.

Struktura half-edge jest powszechnie używaną strukturą w reprezentacji brył, z których buduje się postaci i inne obiekty w filmach animowanych. Zasada modelowania tych tzw. powierzchni **subdivision** polega na iteracyjnym zagęszczaniu ścian poprzez ich podział; zasadę przedstawia seria rysunków:



Bryły z rysunku powyżej były wygenerowane właśnie przy użyciu half-edge. Dlaczego to było takie ważne i wygodne? Tajemnica tkwi w sposobie dzielenia ścian na mniejsze ściany, co zwykle jest dosyć skomplikowanym algorytmem, w którym nowe punkty podziału powstają na ścianach i na krawędziach w zależności od liczby wierzchołków zbiegających się na końcach krawędzi:



Dzięki złożonej strukturze half-edge różnorodne operacje związane z algorytmem podpodziału stają się znacznie bardziej efektywne. Przykładowo, żeby obejść wszystkie krawędzie zbiegające się w jakimś wierzchołku, będącym końcem danej półkrawędzi, należy:

- z danej półkrawędzi przejść na półkrawędź następną
- z tej następnej przejść na przeciwną do niej
- z tej przeciwnej (która "wpada" do wierzchołka) na półkrawędź następną
- z tej następnej znów na przeciwną
- itd., aż wrócimy na półkrawędź pierwotną.

Przejście z jakiejś półkrawędzi na przeciwną lub następną to tylko użycie jednego wskaźnika - błyskawiczna zmiana adresu. Cały obieg krawędzi w wierzchołku zapisuje się więc bardzo zwięźle i wykonuje niezwykle szybko. A spróbujcie to samo wykonać posługując się reprezentacją prostszą, jedną z tych, co na początku rozdziału. To będzie nieporównanie bardziej złożone !