6 Detekcija sudara (engl. collision detection)

- detekcije sudara objekata u pokretu potrebna je jer
  - objekti inače prolaze jedan kroz drugi
  - jako važno kod fizikalni temeljenih simulacija – potreban je odziv na sudar (elastično/plastično) odbijanje objekta
  - povećanje realnosti doživljaja http://alteredqualia.com/xg/examples/animation_physics_ammo.html
  - želimo gibanje jednog objekta po površini drugog (auto) – detekcija kontakta
  - proračun povratne sile (engl. force feedback)
  - izračun osjeta dodira (engl. haptic interaction)

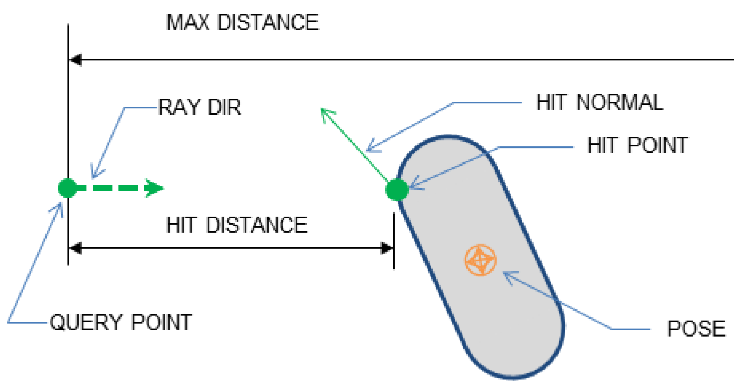
- kod složenih objekata otkrivanje točke sudara vrlo je vremenski zahtijevano
- ovaj proračun je važan kod simulacije strojne obrade površine (glodalica)
  - potrebno je odrediti kada i gdje alat dira površinu
  - da li može pristupiti površini i pod kojim kutom

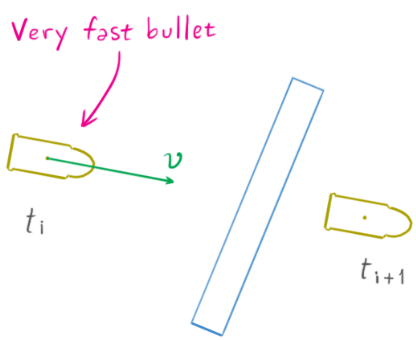
- animacije kod kojih se koristi detekcije sudara
  - kod fizikalno temeljenih animacija - poseban izazov kod konkavnih objekata
    http://chandlerprall.github.io/Physijs/examples/compound.html
  - kod interaktivnog gibanja korisnika (igre, VR)
- fizikalno temeljene simulacije – numerički postupak integracije
  - određivanje pozicije objekta u slijedećem vremenskom trenutku - sudara, proračun je potrebno obaviti prije pomicanja objekta (predviđanje)
  - određivanje sjecišta putanja objekata (da li i kada će se sudariti)
  - https://mrdoob.com/projects/chromeexperiments/google-gravity/

npr. Nvidia PhysX za definiranu udaljenost MaxDistance prati sudare (sweep query)

  - lančani utjecaj na ostale objekte
    http://el-ement.com/etc/oimo/demos/
  - određivanje dubine prodora i minimalne udaljenosti, određivanje kontakta

http://schteppe.github.io/p2.js/demos/collisions.html AABB, Cont





- nedostaci ljudske percepcije kod detekcije sudara
  - ljudi su neuobičajeno loši u određivanju kolizije dva složena objekta, procjenjujemo koliko oko auta ima prostora (na 10 cm)
  - što je više potencijalnih sudara i što se brže dešavaju, teže procjenjujemo
  - u igrama, igrači obično ne vide sami sebe
  - prilično smo loši u predviđanju reakcije sudara
- http://www.magicmgmt.com/gary/oi_whichconnects/
- http://schteppe.github.io/cannon.js/examples/threejs_cloth.html
- osnovni principi
  - prvo se koriste brzi jednostavni testovi kojima ćemo eliminirati potencijalne kolizije (široka faza – broad phase), a zatim vremenski zahtjevni
  - upotreba što jednostavnijih geometrijskih oblika – omeđujućih volumena BV (zastupnika) - za aproksimaciju složenih geometrija (engl. proxy)
  - strukture podataka u kojima je lako odrediti lokalnost i susjedstvo

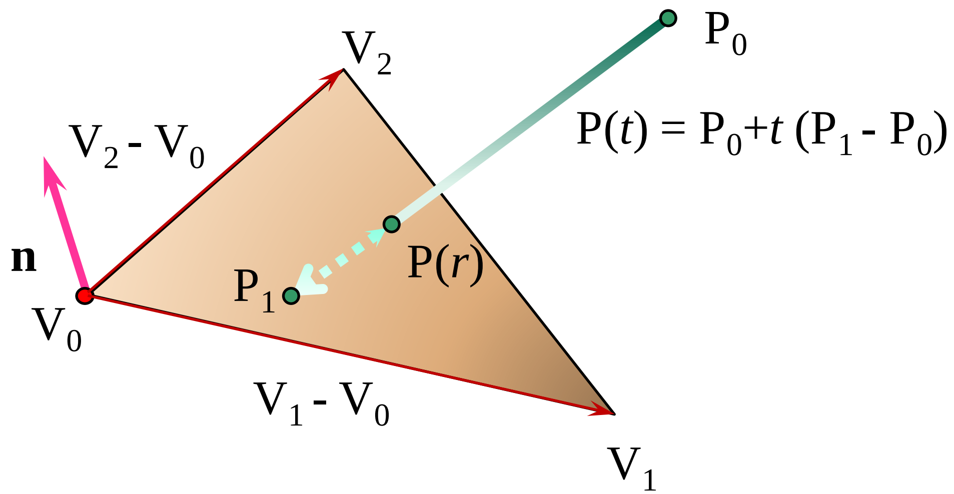
  (engl. acceleration data structures ADS)
  - ako se koristi graf scene, omeđujući volumeni se vežu uz geometrije
  - svojstvo male promjene između susjednih vremenskih trenutaka
- primjena
  - igre – važna je brzina, točnost je sporedna
    http://alteredqualia.com/xg/examples/animation_physics_level.html (Chrome)
  - simulacija robota, glodalice – važna je točnost

6.1 Određivanje sjecišta

- https://stemkoski.github.io/Three.js/Collision-Detection.html A
- ispitivanje da li je vrh jednog objekta unutar drugog
  - postupak ispitivanja da li je točka unutar konveksnog tijela
  - postupak ispitivanja da li je točka unutar konkavnog tijela
  - kugle (sfere) su često korištene, računanje udaljenosti do središte kugle
- ispitivanje sjecišta trokutnih mreža
  - ispitivanje sjecišta brida i trokuta

$$\mathbf{n} = (V_1 - V_0)\times(V_2 - V_0)$$

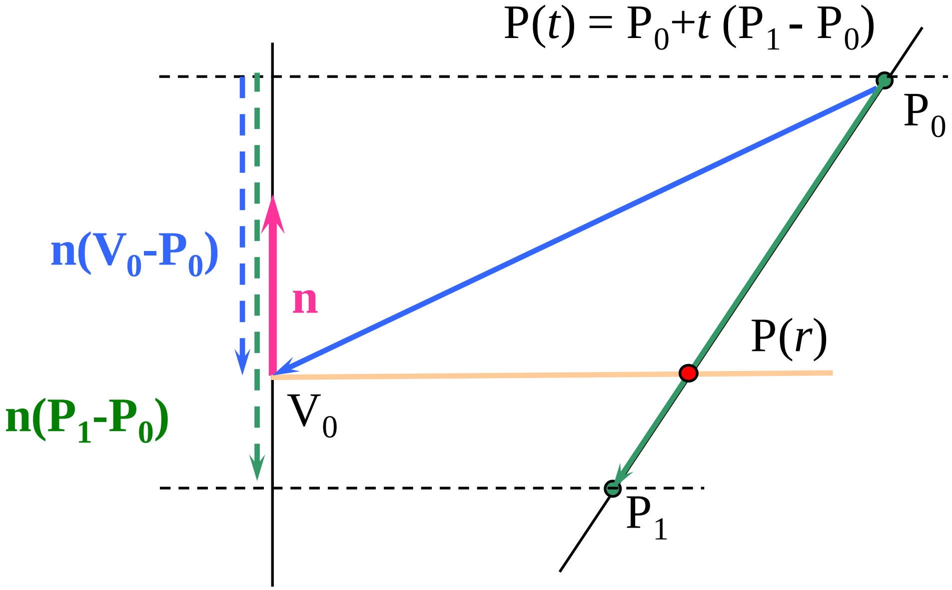
$$r = \frac{\mathbf{n}\cdot\left(V_0 - P_0\right)}{\mathbf{n}\cdot\left(P_1 - P_0\right)}$$



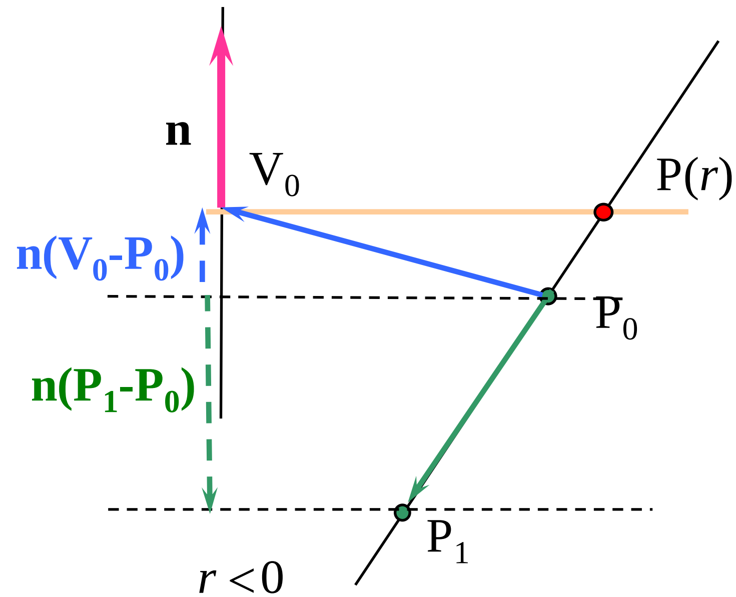
ako je $\mathbf{n}\cdot(P_1-P_0) = 0$ pravac (brida) je paralelan s ravninom (trokuta)

ako je $0 <= r <= 1$ segment siječe trokut u točki $P(r)$

Sjecište



$$r = \frac{\mathbf{n} \cdot (V_0 - P_0)}{\mathbf{n} \cdot (P_1 - P_0)}$$



ako je $\mathbf{n} \cdot (P_1 - P_0) = 0$ pravac (brida) je paralelan s ravninom (trokuta)

ako je $0 <= r <= 1$ segment siječe trokut u točki $P(r)$

– ispitivanje je li točka P(*r*) je unutar trokuta **T**(V_0 V_1 V_2)

parametarska jednadžba ravnine je

$$\boxed{\mathbf{V}(u,v)=(\mathbf{V}_1-\mathbf{V}_0)\,u+(\mathbf{V}_2-\mathbf{V}_0)\,v+\mathbf{V}_0=\vec{v}_{10}\,u+\vec{v}_{20}\,u+\mathbf{V}_0}$$

točka će biti u trokutu **T** ako je (0 <= *u*) i (0 <= *v*) i (*u*+*v* <= 1)

za poznatu točku P(*r*) možemo odrediti *u* i *v*:

$$w=P(r)-V_0$$

$$u=\frac{w\cdot(n\times\vec{v}_{20})}{\vec{v}_{10}\cdot(n\times\vec{v}_{20})}\quad \text{ili}\quad u=\frac{(\vec{v}_{10}\cdot\vec{v}_{20})(w\cdot\vec{v}_{20})-(\vec{v}_{20}\cdot\vec{v}_{20})(w\cdot\vec{v}_{10})}{(\vec{v}_{10}\cdot\vec{v}_{20})^2-(\vec{v}_{10}\cdot\vec{v}_{10})(\vec{v}_{20}\cdot\vec{v}_{20})}$$

$$v=\frac{w\cdot(n\times\vec{v}_{10})}{\vec{v}_{20}\cdot(n\times\vec{v}_{10})}\quad \text{ili}\quad v=\frac{(\vec{v}_{10}\cdot\vec{v}_{20})(w\cdot\vec{v}_{10})-(\vec{v}_{10}\cdot\vec{v}_{10})(w\cdot\vec{v}_{20})}{(\vec{v}_{10}\cdot\vec{v}_{20})^2-(\vec{v}_{10}\cdot\vec{v}_{10})(\vec{v}_{20}\cdot\vec{v}_{20})}$$

gdje je · skalarni i × vektorski produkt

ako je nazivnik nula, radi se o degeneriranom trokutu

– ispitivanje sjecišta dvaju trokuta

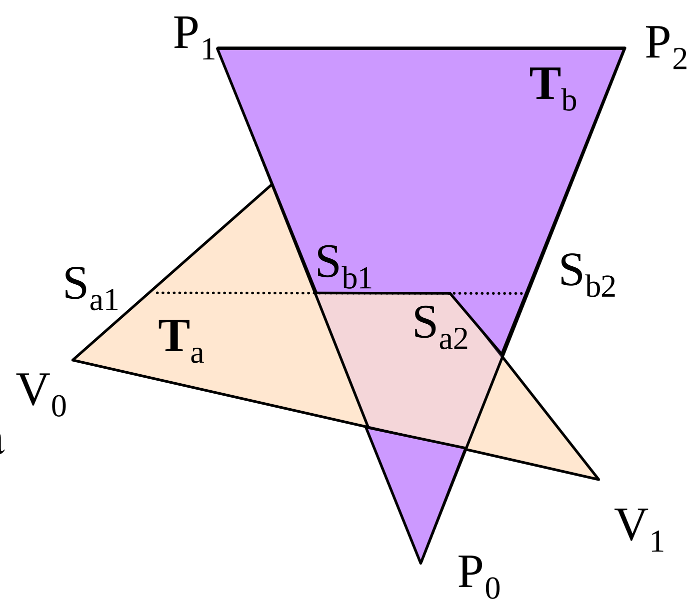
za trokut T_b potrebno je ustanoviti da točke ne leže s iste strane ravnine određene trokutom T_a , neka točka P_0 leži s jedne a P_1 i P_2 s druge strane (pretpostavimo da te ravnine nisu koplanarne)

potrebno je odrediti, prethodnim postupkom sjecište S_{b1} brida P_0P_1 s trokutom T_a i sjecište S_{b2} brida P_0P_2 s trokutom T_a tj. sjecišta koja su unutar $S_b = (S_{b1})$

i na sličan način S_{a1} , S_{a2} tj. $S_a = (S_{a2})$

ako su skupovi prazni trokuti se ne sijeku

inače S_a S_b određuju presjek



brži postupak je

- nalaženje pravca koji je presjek dviju ravnina
- parametarsko određivanje intervala segmenata (samo parametara)
- određivanje preklapanja $S_a = (S_{a1} , S_{a2})$ i $S_b = (S_{b1} , S_{b2})$

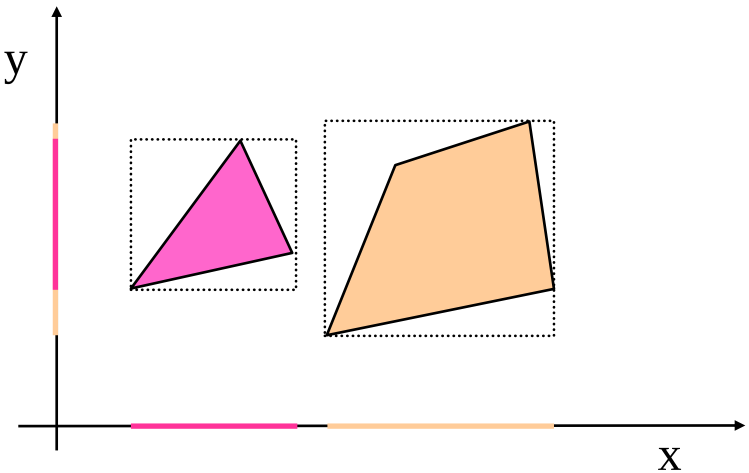
6.2 Optimizacijske strukture podataka

- zbog velike složenosti objekata (dijelova objekata) koriste se
  - omeđujući volumeni (engl. bounding volumes BV)
    - sfere (kugle - jednostavan izračun)
    - kvadri (pravokutnici 2D)
      - poravnati s koordinatnim osima (AABB Axis aligned bounding box)
      - https://stemkoski.github.io/Three.js/Collision-Detection.html (strelice, a,d - pise Hit) http://xeogl.org/examples/#effects_demo_hoverToShowAABB
      - orijentirani (OBB Oriented bounding box) https://threejs.org/examples/webgl_math_obb
    - cilindri (s kuglama - kapsule)
    - trokutne (poligonalne) mreže
  - hijerarhijske strukture omeđujućih konveksnih volumena (BV-trees, BVH) kojima aproksimiramo model
  - različite razine složenosti tijela (LOD level of detail)
  - druge struktura podataka (npr. BSP)

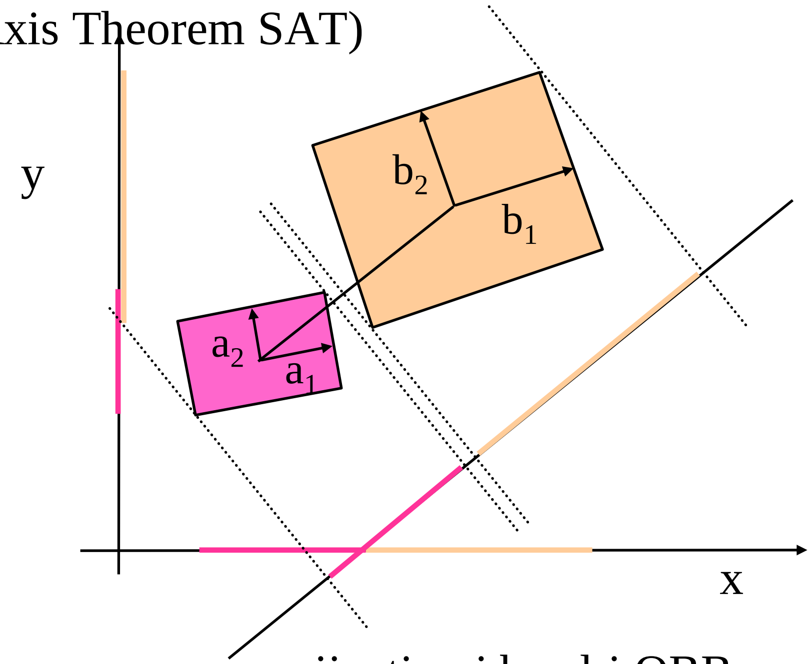
- prednosti
  - postupak ispitivanja uz omeđujući volumen je vremenski manje zahtijevan

- osnovni kompromis u odabiru omeđujućeg volumena je između
  - što manjeg broja poligona
  - što bolja obuhvaćenost objekta https://playground.babylonjs.com/#KQV9SA#0

- omeđujućim volumenima ispitujemo da li se tijela potencijalno sijeku
  - npr. minmaks provjere (uklanjanje skrivenih linija i površina) AABB
    http://mozdevs.github.io/gamedev-js-3d-aabb/api_point.html AABB
    http://el-ement.com/etc/oimo/demos/ T E
  - provjera presjeka orijentiranih kvadara OBB OBB
  - http://chandlerprall.github.io/Physijs/examples/jenga.html
  - ispitivanje se često svodi na smanjenje dimenzije s 3D na 2D i 1D kako bi ustanovili da li se tijela sijeku
  - ispitivanje da li su (konveksni) objekti linearni separabilni, možemo li ih odvojiti ravninom (Separating Axis Theorem SAT)



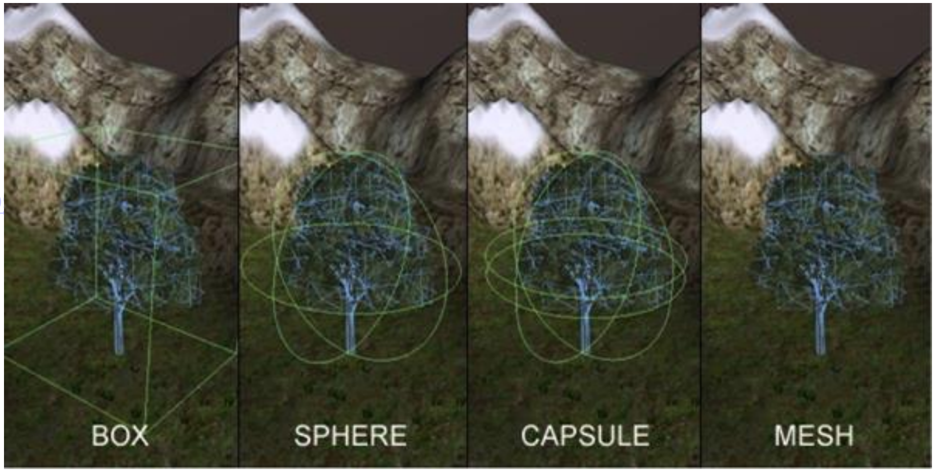
poravnati s koordinatnim osima AABB
„rect-test"(engl. rectangle test)



orijentirani kvadri OBB

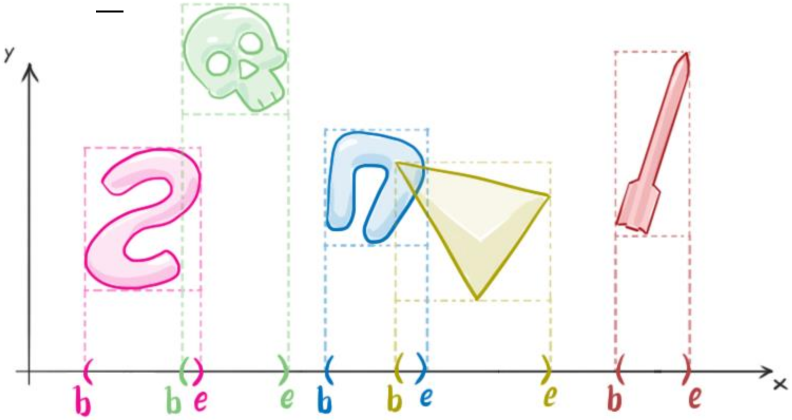
Omeđujući volumeni

- kvadri http://3dflashlo.free.fr/web Option/ShowPl
- sfere
- kapsule
- mreže poligona



Tehnika pometi-pojednostavi (engl. sweep and prune/sort and sweep)

– https://schteppe.github.io/sweep-and-prune/



Stvaranje objekata u posebnim slojevima (engl. layers)

– kako ne bi radili ispitivanje sa svim objektima u sceni radimo provjere samo s objektima u zadanom sloju (npr. ne želimo da eksplozija ne uništi objekte scene već samo objekte neprijatelja) https://threejs.org/examples/webgl_layers.html

- hijerarhija omeđujućih volumena BVH

https://erichlof.github.io/THREE.js-PathTracing-Renderer/BVH_Visualizer.html
http://xeogl.org/examples/#boundaries_scene_aabb

  - sfere
  - orijentirani volumeni OBB

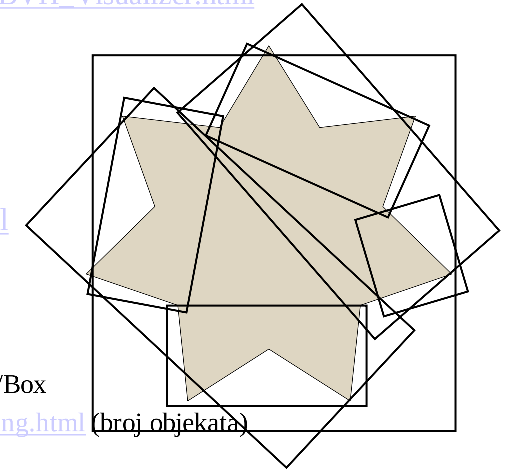
http://schteppe.github.io/cannon.js/demos/ragdoll.html
Rendering/AABB, contacts

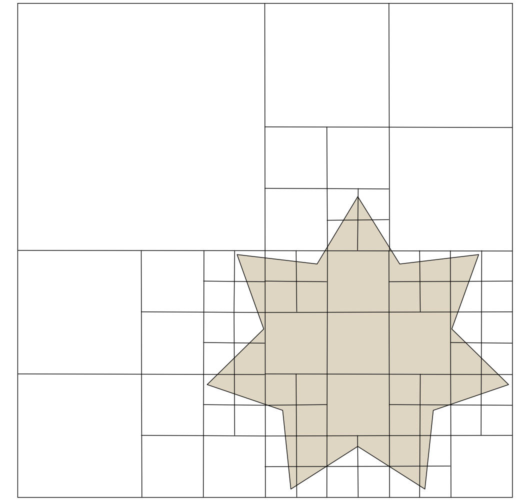
- oktalno stablo

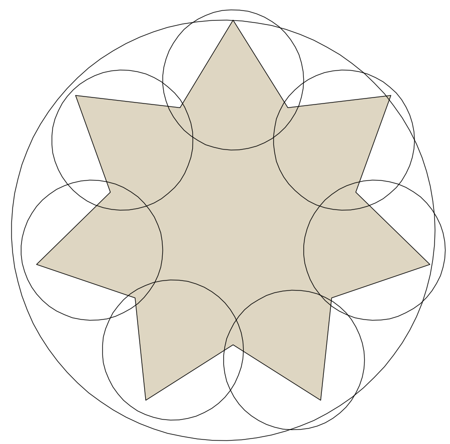
http://potree.org/demo/potree_1.5/examples/lion_laz.html Other/Box
https://softsrc.cc/Public/three.js/examples/webgl_octree_raycasting.html (broj objekata)

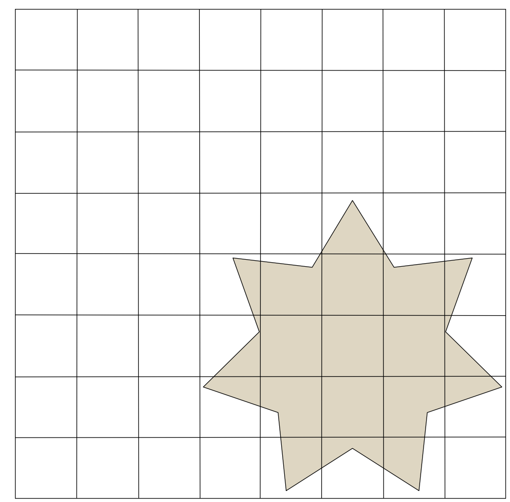
- BSP
- uniformna mreža

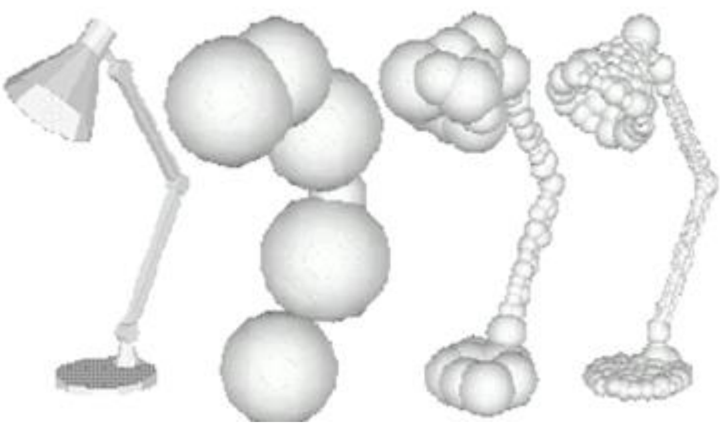
hijerarhijskim postupcima dijelimo samo pod-prostore









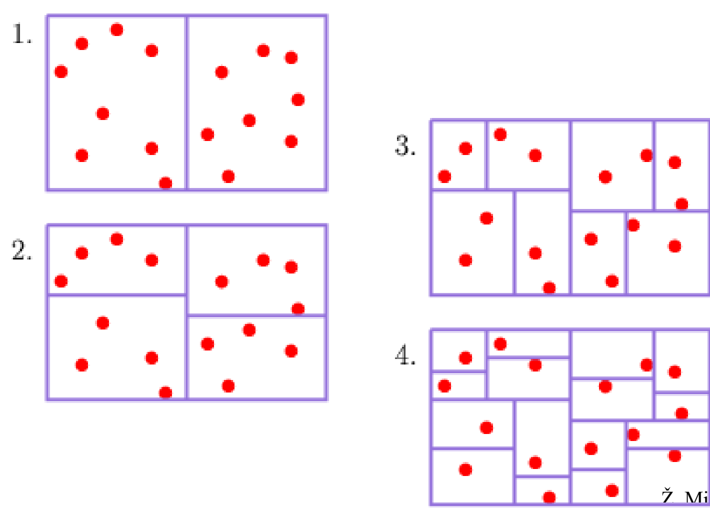


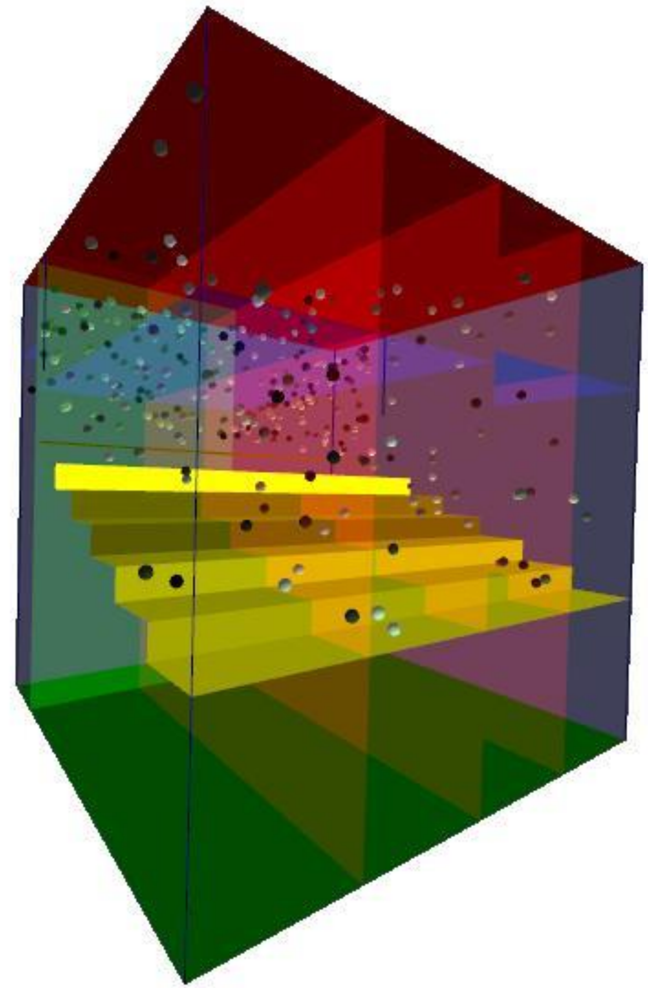
Široka faza ispitivanja kolizije (engl. broad phase)

– primjer – samo-podesivo BSP stablo (kD stablo)

https://va3c.github.io/three.js/examples/#webgl_octree – Oktalno stablo

- broj ispitivanja n(n-1)/2   za 10.000 objekta 50 miliona ispitivanja
- specijalan slučaj BSP stabla kada su ravnine kojima se dijeli prostor poravnate s xy, xz i yz ravninama – potrebna je provjera samo jedne koordinate i veličine objekta
- samo-podesivost – ovisno o gustoći u pojedinom podprostoru definira se veća gustoća podjele http://el-ement.com/etc/oimo/demos/ T

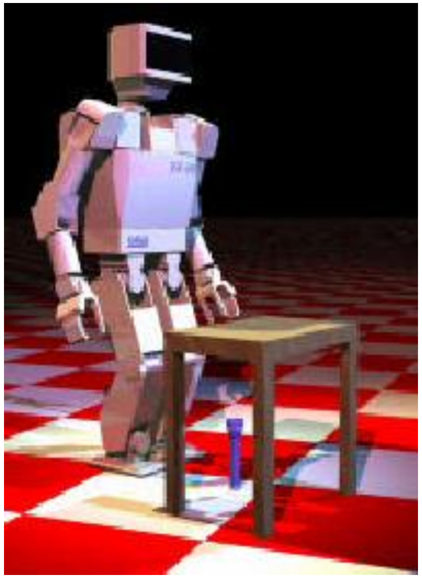


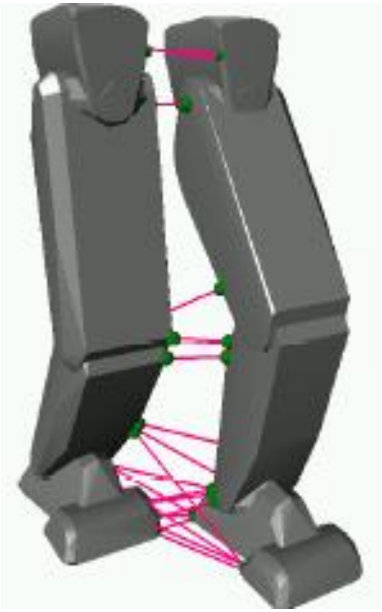


6.3 Određivanje dubine prodora i minimalne udaljenosti

- Određivanje točaka kontakta http://schteppe.github.io/cannon.js/demos/bunny.html
- određivanje dubine prodora konveksnih poliedara (engl. penetration depth)
- za par objekata koji se sijeku to je najkraći vektor za koji jedan od objekta treba translatirati kao bi se objekti došli u položaj dodira

http://www.cs.unc.edu/~geom/Collision_mpeg/kitchen.mpg

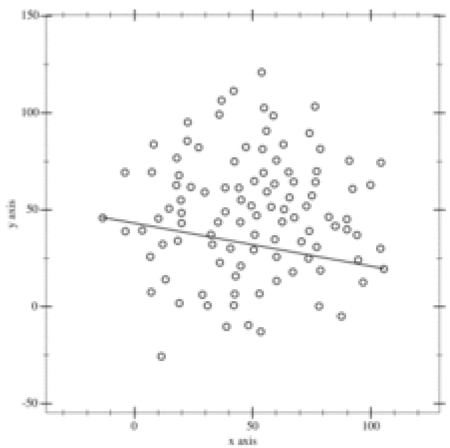




- planiranje putanje objekta

- algoritmi i tehnike koji se koriste
  - određivanje konveksne ljuske
  - Voronoi-evi dijagrami
  - suma (razlika) Minkowskog
  - LOD – algoritmi ugrubljivanja i usitnjavanja objekata

- biblioteke za datekciju kolizije, primjeri
  - http://gamma.web.unc.edu/research/collision/
  - www.bulletphysics.org/



Quickhull $O(n \log n)$

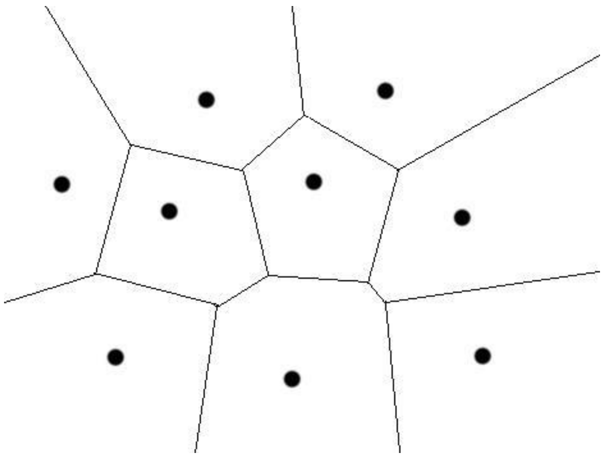
- dijagram Voronoi (computational geometry)

  – dan je skup S s n točaka u ravnini R^2, za svaku točku p_i iz skupa S postoji skup točaka ravnine koje su bliže točki p_i od bilo koje druge točke iz skupa S. Skup takvih točaka zove se Voronoi-ev poligon. Skup n Voronoi-evih poligona za n zadanih točaka čine Voronoi-ev dijagram.

  – drugim riječima, podjela ravnine u regije točaka koje su najbliže pojedinoj točki p_i iz skupa S.

  – http://bl.ocks.org/mbostock/6675193

  – http://alexbeutel.com/webgl/voronoi.html



  – za dvije točke možemo odrediti pravac
koji predstavlja skup točaka jednako udaljenih
od te dvije točke

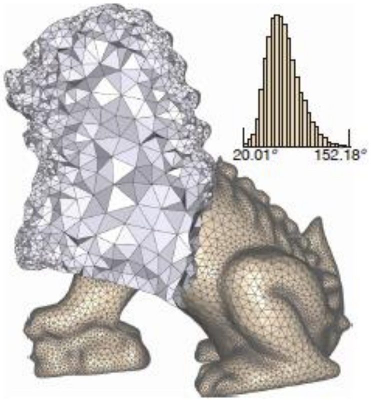
  – sjecište dva pravca je jednako udaljeno
od tri promatrane točke (opisana kružnica trokuta)

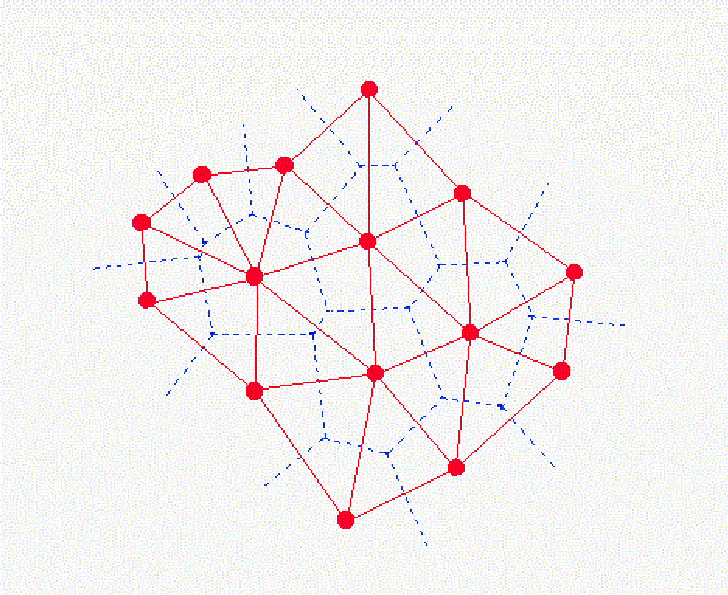
http://www.senchalabs.org/philogl/PhiloGL/examples/voronoi/ Dvoklik

Primjena:

Select http://bl.ocks.org/njvack/1405439

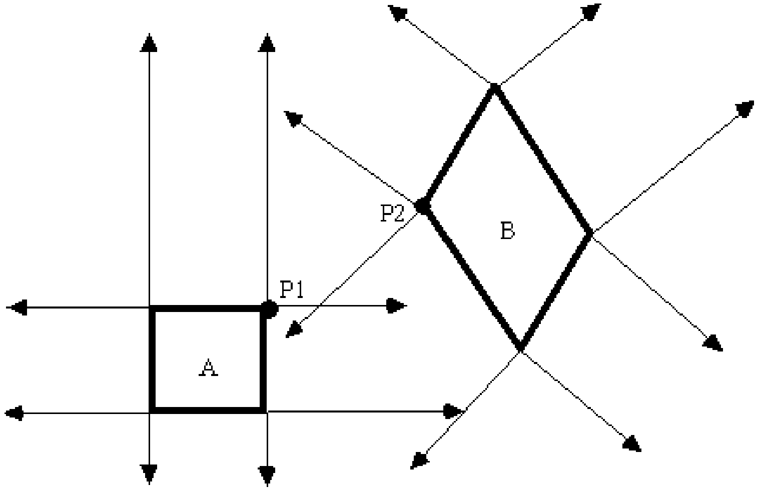
- triangulacija Delaunay
  - jedinstvena triangulacija načinjena tako da ni jedna točka iz skupa *S* ne leži u opisanoj kružnici drugog trokuta
  - simetrale stranica trokuta određuju opisanu kružnicu trokuta
  - 3D Voronoi/Delaunay/konveksna ljuska
  - http://callumprentice.github.io/apps/delaunay/index.html#
  - https://www.shadertoy.com/view/ldV3Wc
  - https://www.yuichiroharai.com/wgl/10_delaunay/





- Voronoieve regije i dijagram za konveksne poligone
    - dan je skup poligona, koji se sastoje od primitiva, primitive su:
        - točka
        - brid
    - traže se podjela na regije,
    - pojedinu regiju čine točke najbliže zadanim primitivama

Objekti A, B i Voronoi-eve regije
P_1 i P_2 su najbliže za A, B akko
P_1 je u Voronoi-evoj regiji od P_2, i obrnuto



Primjer primjene:

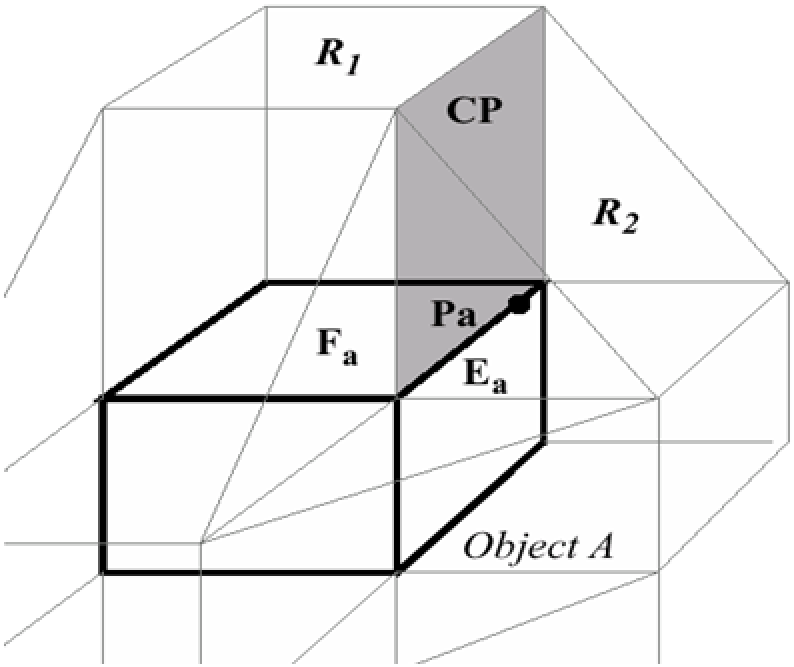
Particioniranje objekta - lomljenje: http://schteppe.github.io/voronoi-cube/ https://demo.marpi.pl/explosion/
https://yomboprime.github.io/ConvexBreakSound3D/webgl_physics_convex_break.html

Pokrivenost aerodromima: http://mbostock.github.io/d3/talk/20111116/airports-all.html

- Voronoi-eve regije i dijagram konveksnih poliedara
  - proširenje koncepta na tri dimenzije znači određivanje Voronoi-evih regija tj. skupova točaka 3D prostora
  - dan je skup konveksnih poliedara, koji se sastoje od primitiva, primitive su:
    - točka
    - brid
    - poligon objekta

    http://jimmyland.github.io/voro/

    - pomoć u kreiranju objekata
  - regiju čini skup točaka najbližih pojedinoj primitivi
  - ispitivanje se svodi na provjeru da li se točka nalazi u konveksnom poliedru



  - problem d-dimenzijskog linearnog programiranja

Suma/razlika Minkowskog konveksnih objekata

- suma Minkowskog (A, B) = { a + b | a ∈A, b ∈ B}
- razlika Minkowskog (A, B) = { a - b | a ∈A, b ∈ B}

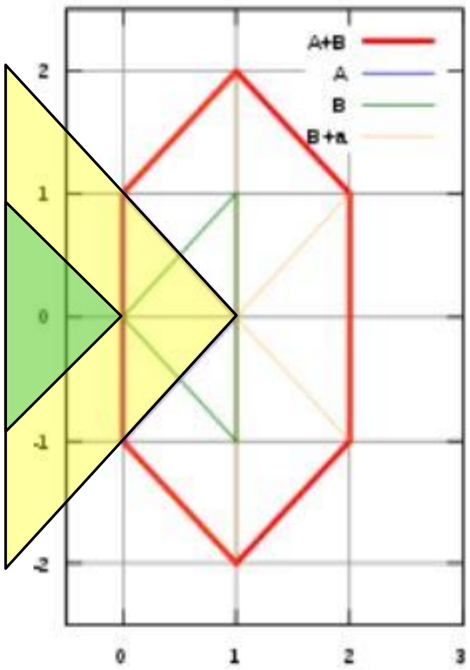
Npr. dva trokuta: (plavi) $A = \{ (1, 0), (0, 1), (0, -1)\}$, (zeleni) $B = \{ (0, 0), (1, 1), (1, -1)\}$,

suma Minkowskog je konveksni objekt (crveno):

$A + B = \{ (1, 0), (2, 1), (2, -1),$ // zbrojimo svaku točku iz A

$(0, 1), (1, 2), (1, 0),$ // sa svakom točkom iz B

$(0, -1), (1, 0), (1, -2)\}$



razlika Minkowskog – prvo odredimo (-B)
tako da promijenimo predznake

to je centralno simetrična slika obzirom na ishodište i
napravimo sumu A+ (-B)

(-B) = { (0, 0), (-1, -1), (-1, 1)},

A + (-B) = { (1, 0), (0, -1), (0, 1),

(0, 1), (-1, 0), (-1, 2),

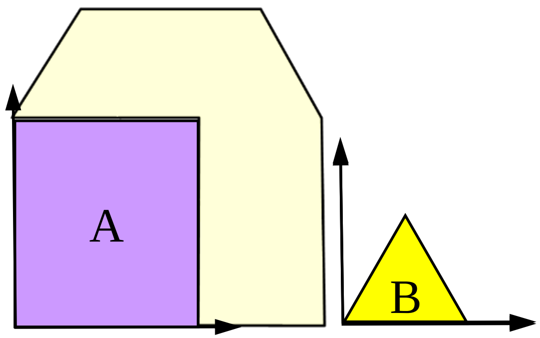
(0, −1), (-1, -2), (-1, 0)}

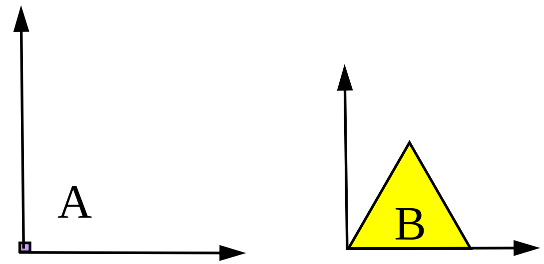
Ako ovaj konveksni oblik sadrži **ishodište** objekti su u koliziji!!!

Suma Minkowskog – sumiramo svaku točku objekta A sa svakom točkom objekta B

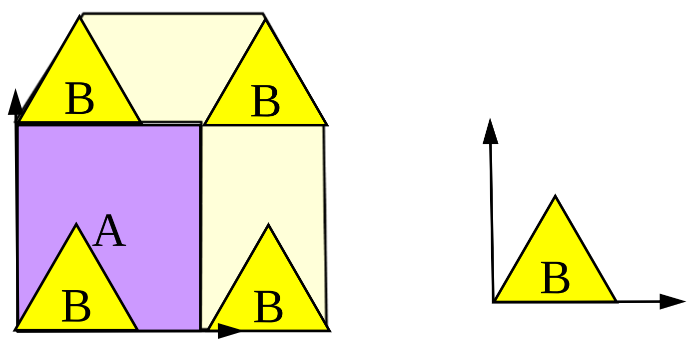
$$A + B = \{a + b \mid a \in A,\ b \in B\}$$

Npr. Neka su objekti A i B i objekt B „provozamo” po rubu objekta A:



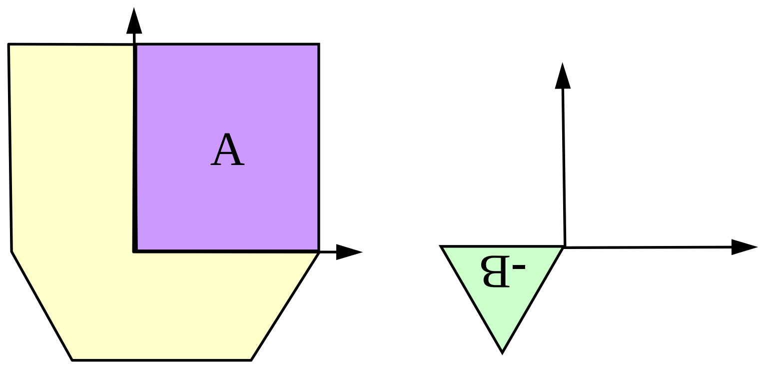


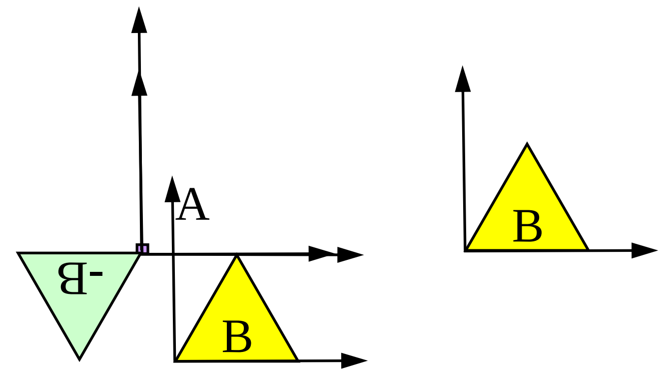
Kada su A i B konveksni objekti možemo za svaki vrh objekta A zbrojiti vrhove objekta B. Konveksna ljuska tako dobivenih točaka je Suma Minkowskog A + B.



Razlika Minkowskog - zrcalimo objekt B obzirom na ishodište koordinatnog sustava i sumiramo s objektom A

Npr. Neka su objekti A i B:





Koristi se pri određivanju prostora u kojem se objekt B može kretati a da nije u koliziji s A.

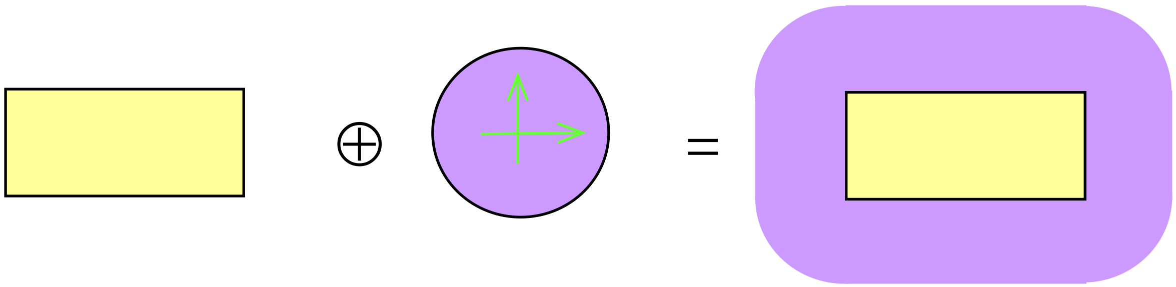
U ovom primjeru nas zanima prostor oko objekta A u kojem se objekt B može kretati a da ne dođe do kolizije – prati se samo ishodište objekta B i dobivena razlika Minkowsog.

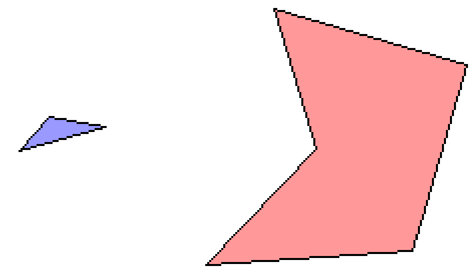
Određivanje dubine prodora: Za ishodište objekta B kada je unutar područja dobivenog razlikom Minkowskog treba naći najbliži brid obzirom na ishodište
(time dobijemo vektor pomaka potreban da izađemo iz kolizije)

http://www.cs.tau.ac.il/~danha/video/EMINK.mpg

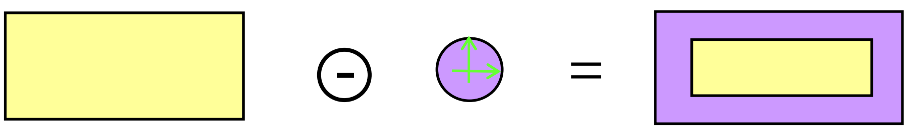
Matematičkom morfološki operatori

- rastezanja $A \oplus B$ - za slučaj kruga odgovara sumi Minkowsokog





- erozije (nagrizanja)



- primjena
  - određivanje putanje robota (prostor gdje se može kretati da ne kolidira) http://sunag.github.io/sea3d/Examples/Programmer/Three.JS/demos/spider_cloud.html
  - određivanje dubine prodora
  - određivanje minimalnog puta koji treba robot prijeći od A do B na poligonu s preprekama Klavir https://www.blend4web.com/apps/code_snippets/code_snippets.html?scene=pathfinding

http://math.berkeley.edu/~sethian/2006/Applets/Robotics/java_files_robotic_illegal/robotic_coarse.html

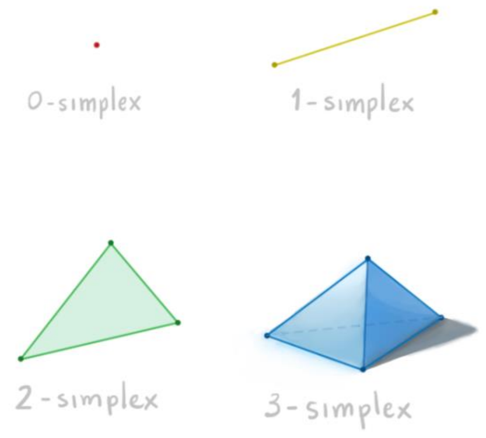
http://nickjanssen.github.io/PatrolJS/demo/demo.html

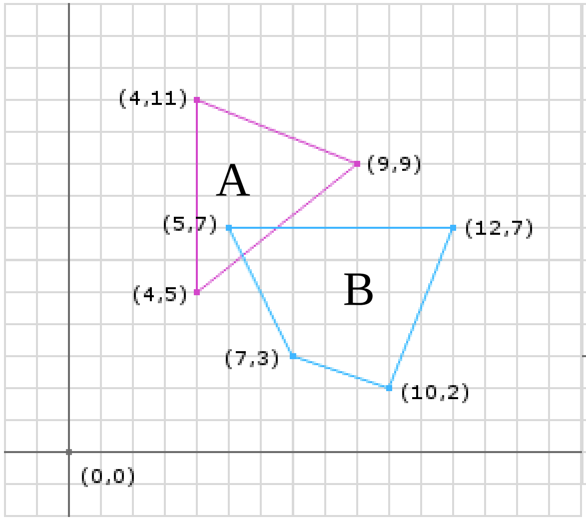
http://qiao.github.io/PathFinding.js/visual/

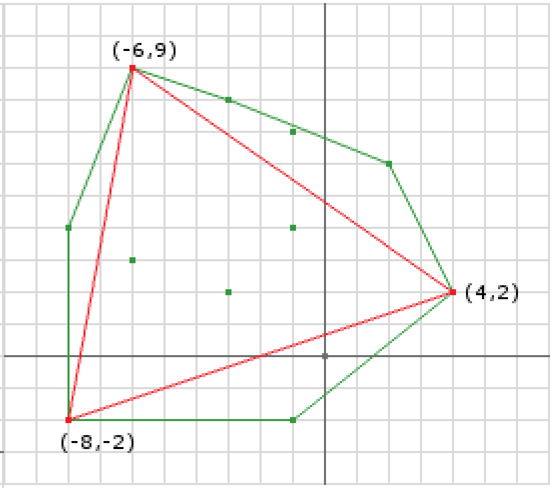
GJK algoritam (Gilbert-Johnson-Keerthi)

- koristi činjenicu da razlika Minkowskog sadrži ishodište – točku O (0, 0) ako su objekti u koliziji
- suma/razlika Minkowskog dva konveksna objekta je konveksan objekt,
- cilj je odrediti da li razlika Minkowskog sadrži ishodište bez da određujemo sumu/razliku već iz poznavanja točaka objekta A i B.
- algoritam radi iterativno tako da se odredi proizvoljan trokut (engl. Simplex) unutar razlike Minkowskog i provjerava se da li on sadrži ishodište
- vrhovi simpleksa se biraju tako da se u iterativnom postupku odabir *usmjerava* prema ishodištu

Simpleks (2D-trokut):





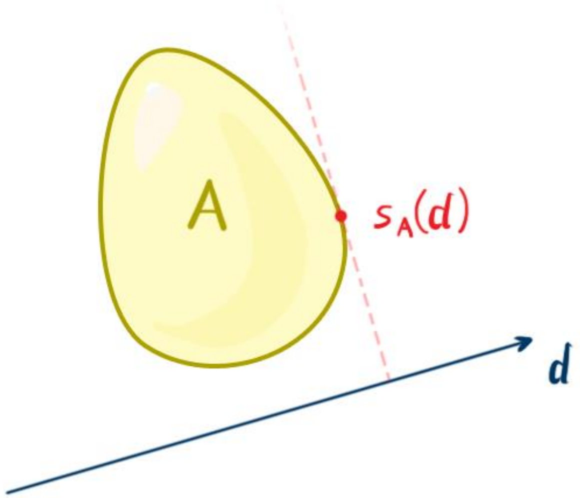


GJK algoritam (Gilbert-Johnson-Keerthi)

Usmjeravanje odabira – funkcija *support*(A, **d**)

- nalazi najudaljeniju točku skupa A u smjeru **d**

(npr. preko skalarnog produkta točaka-vektora iz ishodišta s vektorom **d**)



Usmjeravanje odabira – funkcija *support*(A, B, **d**)

d - smjer (vektor)

- nalazi najdalju točku u skupu A u smjeru **d**, i najdalju točku u skupu B u smjeru **–d** i oduzmemo ih.

GJK primjer:

Odaberemo neki **d** = (1, 0) // pogodan odabir bi bio vektor između središta objekata A i B

p_1 = (9, 9)=*support*(A, **d**);

p_2 = (5, 7) =*support*(B, -**d**);

$p_3 = p_1 - p_2$ = (4, 2)= *support*(A, B, **d**);

Biramo smjer suprotno od prethodnog tj. **d** = (-1, 0)

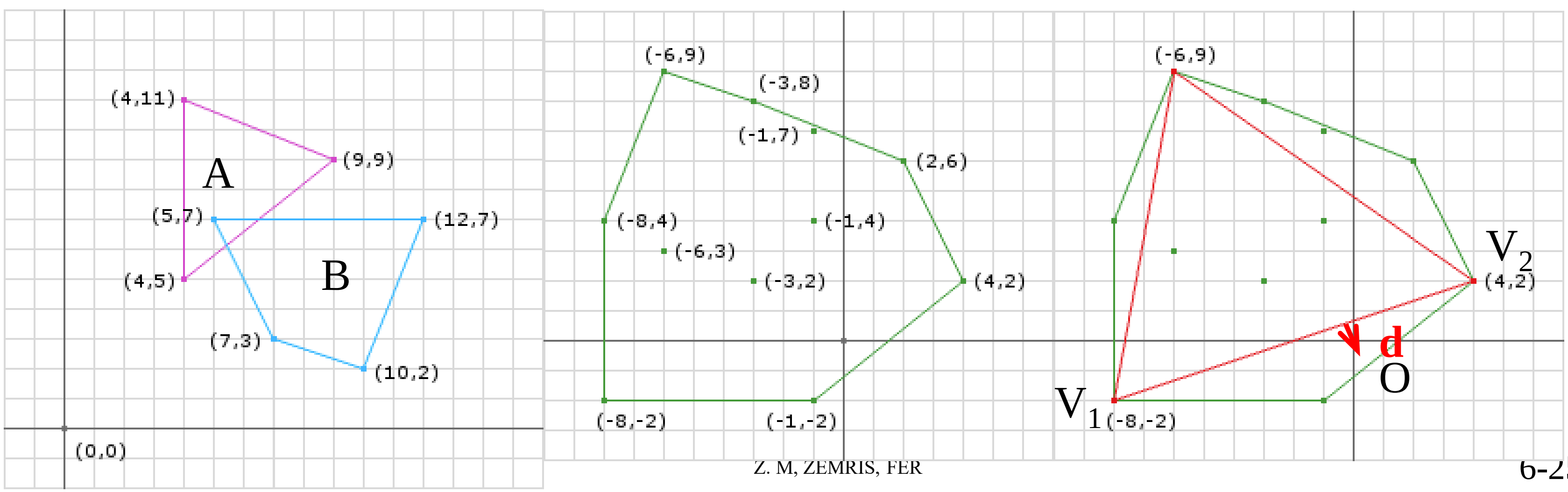
p_1 = (4, 5); ili (4, 11) – dao bi isto ispravno rješenje

p_2 = (12, 7);

$p_3 = p_1 - p_2$ = (-8, -2);

Za **d** = (0, 1) bi dobili treći vrh trokuta (-6, 9) no za **d** = (0, -1) bi trokut obuhvatio ishodište.

Treću točku biramo tako da **d** usmjerimo okomito na spojnicu prve 2 točke prema ishodištu odnosno **d** = $(V_2-V_1) \times ((O-V_1)\times(V_2-V_1))$. Postupak se iterativno ponavlja.



GJK algoritam (Gilbert-Johnson-Keerthi)

Neka je plavo tijelo razlika Minkovskog A-B.

Mi želimo naći točku plavog tijela koja je najbliža ishodištu O.

U skupu W={} čuvamo potencijalne kandidate – točke koje su najbliže ishodištu. Postupak iterativno ponavljamo. Kada je kut između potencijalnih kandidata dovoljno mali algoritam zaustavljamo, ili ako je O unutar plavog skupa tada su objekti u koliziji.

U našem primjeru zeleni vektor **–d** usmjerava odabir

- nađemo w_0=*support*(A, **d**)
- okomito na vektor d je w_1=*support*(A, ⊥**d**)

(želimo da naš simpleks bude rastegnut pa je zato ⊥)

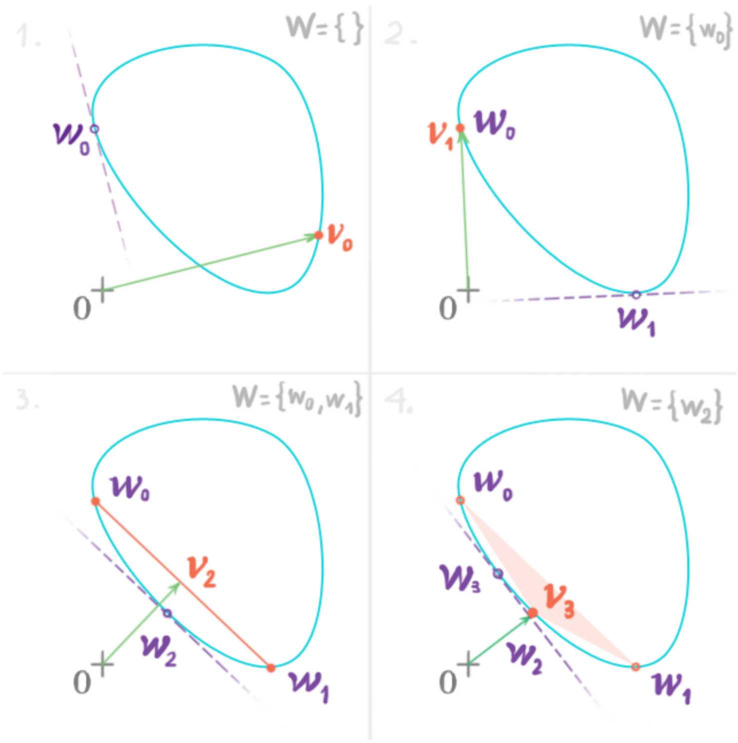
- na spojnicu $w_0 w_1$ odredimo smjer d_1 prema ishodištu

w_2=*support*(A, $\mathbf{d_1}$)

- provjeravamo gdje je ishodište, iteriramo

Rezultat tj. kut između vektora Ow_2 i Ow_3 treba biti <ε

→ minimalna udaljenost objekata B i A i smjer.



http://schteppe.github.io/cannon.js/demos/ragdoll.html Rend/cont

Polja udaljenosti (engl. signed distance fields SDF)

- za svaki objekt načinimo volumen podataka npr. 100 x 100 x 100 ili za scenu i više
- unutar volumena za svaki *x, y, z* odredimo minimalnu udaljenost (i smjer-gradijent) do površine objekta
  - kod objekata zadanih poligonima možemo načiniti Voronoi-eve regije pa udaljenost određujemo do poligona, brida, vrha objekta
- kada dođe do prodora jednog volumena u drugi, za svaku točku jednog objekta provjeravamo u polju udaljenosti drugog objekta položaj te točke (je li unutar)
- polja udaljenosti su specijalan slučaj funkcije skupa razina (izofunkcije, level set)
- http://cagd-applets.webarchiv.kit.edu/mocca/html/noplugin/IntBSpline/AppDifferentialGeometry/index.html
- https://www.shadertoy.com/view/Xtl3Wj