PV251 Vizualizace

Jaro 2016

Výukový materiál

---

9. přednáška: Interakční techniky

V této přednášce se zaměříme na algoritmy a implementační detaily související s koncepty interakce popsány v minulé přednášce. Zmíníme příklady pro každý z uvedených interakčních prostorů a poté poskytneme návod pro implementaci některých typů uživatelské interakce. Většinu uvedených algoritmů je možné použít pro interakci v násobných prostorech.

Prostor obrazovky

Distorze (deformace) v prostoru obrazovky je běžným nástrojem pro poskytnutí focus+context techniky uživateli (tj. detail části, zbytek schematicky). Podíváme se blíže na jeden z typických příkladů této distorze: na rybí oko. Implementace rybího oka je poměrně přímočará. Je nutné specifikovat středový bod (c_x, c_y) transformace, poloměr čočky (lupy) r_l a velikost vychýlení (deflexe) d. Poté dochází k transformaci souřadnic obrázku do polárních souřadnic a to relativně ke středovému bodu. Efekt lupy je získán poměrně jednoduchou transformací aplikovanou v rámci poloměru r_l.

Jedna z populárních transformací tohoto typu je zadána vzorcem:

$$r_{new} = s \log(1 + d(r_{old}))$$

kde

$$s = \frac{r_l}{\log(1 + d * r_l)}$$

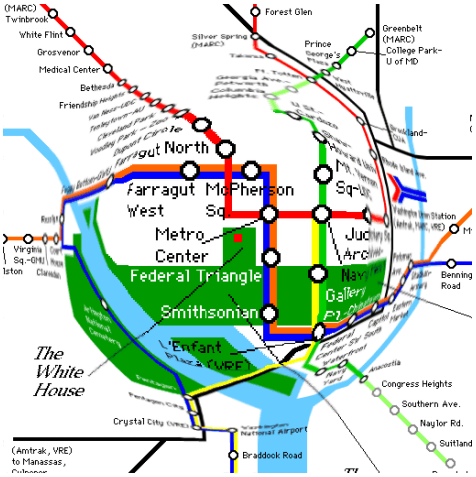
Tento vzorec zajišťuje, že poloměr bodů nacházejících se na okraji lupy je zachován na své původní hodnotě. Nyní si uvedeme pseudokód celého algoritmu:

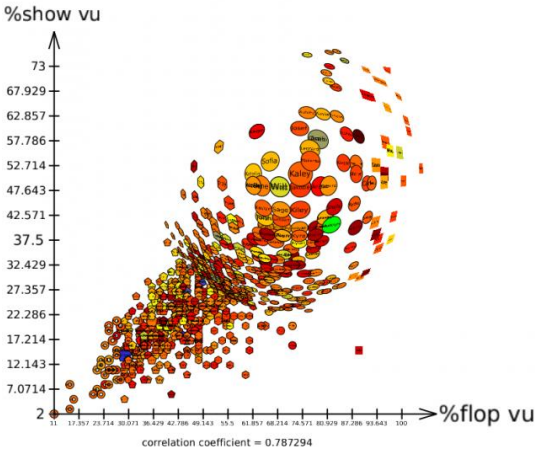
1. Vyčistíme výstupní obrázek.
2. Pro každý pixel vstupního obrázku:
    1. Spočteme odpovídající polární souřadnice.
    2. Pokud je poloměr menší, než je poloměr lupy:
        1. Spočteme nový poloměr r_{new}.

2. Získáme barvu tohoto místa z původního obrázku.
3. Nastavíme tuto barvu jako barvu pixelu ve výstupním obrázku.

3. Jinak nastavíme výsledný pixel obrázku na stejnou hodnotu, jakou má v původním obrázku.

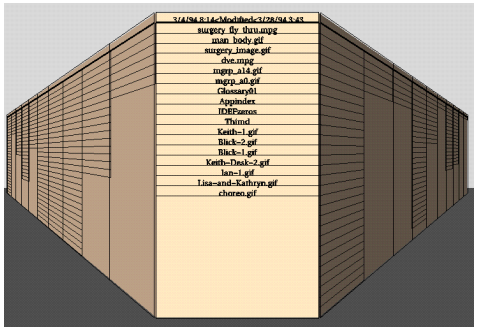
Podle typu transformace použité pro distorzi v prostoru obrazovky se musíme příslušně vypořádat s děrami nebo naopak s překrytím pixelů, ke kterému při transformaci dochází. Překrývání pixelů nezpůsobuje tak velké problémy jako díry. Přesto jsou překrývající se pixely často v konkrétních implementacích průměrovány. Díry je nutné vyřešit pomocí interpolace. Zde záleží na typu použité lupy – pro vizualizaci textu se pro interpolaci nejčastěji používá po částech lineární funkce s „rovnou“ (flat) středovou částí, aby bylo možné text v této části bez problémů číst.



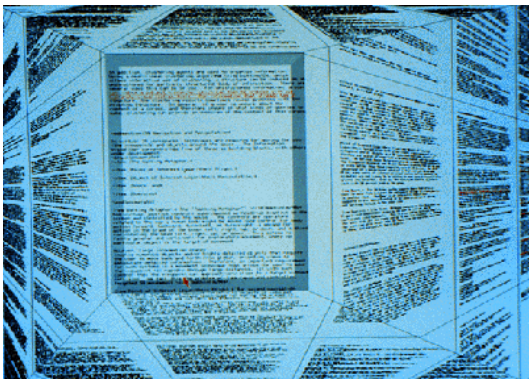


Prostor objektu (3D surfaces)

Jednou z metod pro navigaci ve vizualizaci velkého počtu dokumentů a dat jsou takzvané **perspektivní stěny** (perspective walls). Tento přístup zobrazuje jeden panel pohledu na povrch objektu umístěný ortogonálně ke směru pohledu a ostatní panely jsou orientovány takovým způsobem, že mizí se vzdáleností a to způsobem definovaným pomocí perspektivní deformace.



Zjednodušená verze perspektivní stěny může být vytvořena tak, že přední stěna implementuje horizontální škálování určité části mapovaného 2D obrázku, zatímco sousední segmenty jsou podrobeny horizontálnímu a vertikálnímu škálování, které je úměrné jejich vzdálenosti ke hraně přední stěny. Navíc je na tyto segmenty aplikováno zkosení.



Tedy pokud jsou levá, střední a pravá sekce původního obrázku, který má být vykreslen na perspektivní stěnu, ohraničeny pomocí (x_0, x_{left}, x_{right}, x_1) a levý, střední a pravý panel výsledného obrázku je definován jako (X_0, X_{left}, X_{right}, X_1), pak aplikovaná transformace je následující:

– pro $x < x_{left}$:

$$x' = X_0 + (x - x_0) * \frac{(X_{left} - X_0)}{(x_{left} - x_0)}$$

$$y' = (X_{left} - x') + y\left(1 - \frac{(X_{left} - x')}{(X_{left} - X_0)}\right)$$

– pro $x_{left} \le x < x_{right}$:

$$x' = X_{left} + (x - x_{left}) * \frac{(X_{right} - X_{left})}{(x_{right} - x_{left})}$$

$$y' = y$$

– pro $x \ge x_{right}$:

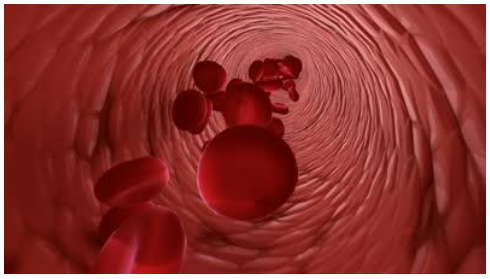
$$x' = X_{right} + (x - x_{right}) * \frac{(X_1 - X_{right})}{(x_1 - x_{right})}$$

$$y' = (x' - X_{right}) + y\left(1 - \frac{(x' - X_{right})}{(X_1 - X_{right})}\right)$$

Při použití perspektivní stěny s ní může uživatel interagovat sekvenčním procházením „stránek“ (dopředu i dozadu). Dále je možné využívat indexy pro přímý přístup (skákání) do oblasti zájmu – často je toto implementováno jako záložka vyčnívající nahoře na stránce na začátku každé sekce.

Další metody prozkoumávání/navigace

Další formy prozkoumávání/navigace mají podobné potřeby. Například průlet skrz datovou sadu reprezentující lidské srdce nebo trávicí systém požaduje ovládací prvky pro určení směru a rychlosti pohybu a rovněž pro určení field of view (oblasti pohledu). Pokud zůstaneme u medicínské vizualizace, pak například průlet skrz cévy řízený uživatelem může být při určité rychlosti problematický – pokud si uživatel přeje zůstat uvnitř cévy. Je to velmi podobné jako jakýkoliv 3D letecký či jiný simulátor, kdy je při vyšších rychlostech obtížné se vyhnout kolizím.



Pro vizualizaci průletu skrz data tedy může být jedním z požadavků možnost automatického nastavení cesty průletu a dokonce můžeme uživateli přikázat, aby udržoval fixní orientaci pohledu (zabráníme otáčení). Pro případ cévního systému tento přístup vynucuje nalezení středového bodu každého kusu (slice) cévy a spojení těchto pozic mezi snímky. Toto však může vést k vytvoření poměrně kostrbaté průletové cesty, protože jednotlivé kusy v cévě nejsou vždy perfektně zarovnány. Navíc segmentace na hranicích cév může zavést nepřesnosti mezi snímky. Proto je často nutné do tohoto řešení zahrnout operaci vyhlazování.

Vyhlazování cesty mezi diskrétními body je možné provést několika různými způsoby. Nejjednodušší metodou je jeden nebo více průchodů algoritmu pro **průměrování sousedů**, kdy je každý bod na cestě nahrazen průměrnou hodnotou získanou z určitého množství bodů před a za tímto bodem. Nejjednodušší příklady jsou následující:

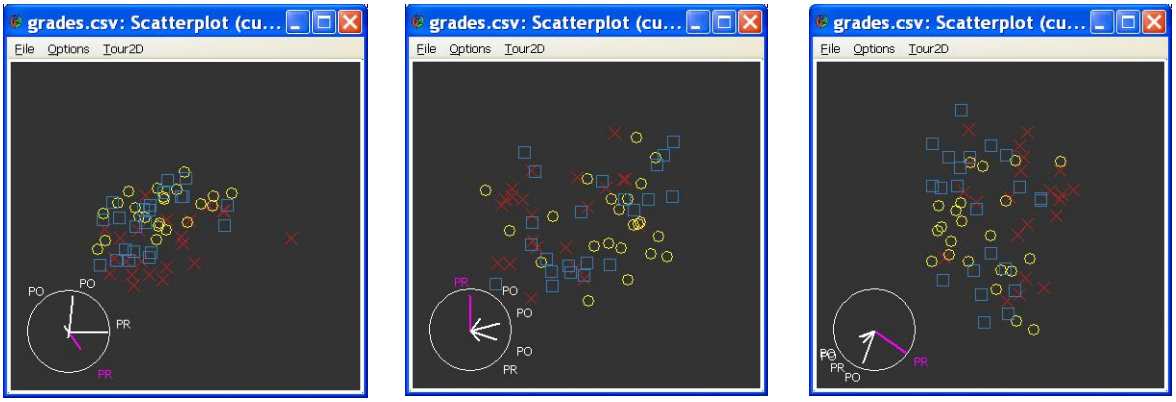
$$p_i^{'} = (p_{i-1} + p_{i+1})/2$$

$$p_i^{'} = (p_{i-1} + p_i + p_{i+1})/3$$

Použití většího počtu sousedů nebo násobné průchody průměrováním zvyšují vyhlazení cest, ale zároveň se zvyšuje riziko, že „vystoupíme“ z cévy ven – obzvláště ve velmi zakřivených částech. Toto můžeme zmírnit přidáním různých omezení, jako zavedení odpudivých sil u hranic cév. Ovšem zaplatíme za to vyšší cenou za zpracování vyhlazování.

Dalším možným přístupem je napasování bodů na parametrickou křivku, která pak může být použita pro generování cesty o libovolném počtu kroků. Příkladem takovéto křivky je Bézierova křivka nebo B-spline. Ve více zakřivených částech můžeme jednoduše přidat více kontrolních bodů.

Rovněž byly prozkoumávány plně automatizované techniky. Jednou z populárních metod pro multivariate statistickou vizualizaci je tzv. **Asimov's grand tour** (http://www.slac.stanford.edu/cgi-wrap/getdoc/slac-pub-3211.pdf), která generuje sekvenci projekcí dat o vysoké dimenzi do 2D nebo 3D, aby je bylo možné zobrazit. Cesta je navržena takovým způsobem, že každou možnou projekci (omezena pouze velikostí kroku) je možné eventuálně navštívit. Navíc sousední projekce se liší v pohledových parametrech pouze minimálně, proto je přechod mezi nimi vnímán jako hladký.



Prostor dat (Multivariate Data Values)

Transformace v prostoru dat jsou velmi běžnými technikami v analýze a vizualizaci dat. Některé typické funkce jsou:

- Škálování a translace, abychom se dostali na žádaný rozsah hodnot
- Exponenciální a logaritmické škálování pro rozšíření/kompresi rozložení dat
- Sinusoidy pro studování cyklických závislostí
- Transformace absolutní hodnoty
- Negace hodnot, kdy nízké hodnoty se stanou velkými a naopak

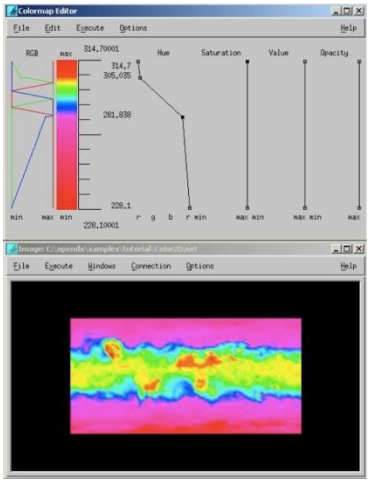
Klíčovým požadavkem pro tento i jiné typy transformace je skutečnost, že uživatel byl informován o tom, že data byla nějakým způsobem transformována. Dále je nutné jasně označit jednotlivé osy. Rozšíření vizualizace o poznámku o tom, jaké transformace byly použity, výrazně redukuje riziko špatné interpretace.

Další důležitým požadavkem je transformace rozsahu hodnot výsledných dat takovým způsobem, aby spadaly do rozsahu akceptovatelného grafickými entitami a dalšími renderovanými atributy. Pokud je tato transformace aplikována chybně, mohou být ve výsledku hodnoty namapovány mimo obrazovku nebo může dojít k jiným nežádoucím artefaktům.

Pravděpodobně nejpoužívanější interakce v prostoru atributů představují modifikace barevných atributů a atributů průhlednosti. Byla navržena řada technik, které se snaží o lepší využití barevného prostoru a lepší vnímání vlastností zobrazovaných dat. Například řízením kontrastu a jasu můžeme zvýraznit určitou oblast dat, čímž můžeme přilákat pozornost uživatele, který data analyzuje. Navíc mu usnadníme detekci vlastností, klasifikaci a měření dat. Příklad je na obrázku, kdy změnou kontrastu a jasu zvýrazňujeme jisté vlastnosti.

Interaktivní nástroje pro specifikování a modifikaci přenosové funkce (transfer function) se nejčastěji používají při renderování objemu (volume rendering) pro kontrolu barvy a průhlednosti, čímž dokážeme zvýraznit různé struktury uvnitř objemových dat. V nejjednodušší formě se tyto nástroje skládají z vizuální reprezentace funkce vykreslující hodnoty dat (horizontální osa) a z průhlednosti či barevné komponenty (HSV – hue, saturation, value) – viz obrázek, který ukazuje prudké změny v sytosti při malém rozsahu datových hodnot.

Uživatel může vkládat a modifikovat kontrolní body, výsledná funkce má tvar po částech lineárního grafu procházejícího skrz sousední kontrolní body.



Nevýhodou je skutečnost, že i malé změny v přenosové funkci mohou vést k výrazným změnám ve vizualizaci. Dalším problémem této strategie je zakládání barvy nebo průhlednosti striktně na datových hodnotách – to může vést k vizuálním artefaktům způsobeným šumem nebo variabilitou uvnitř dat.

Možným řešením tohoto problému je využití i jiných charakteristik dat. Příkladem je využití první a druhé derivace, na jejichž základě se vytváří 3D histogram.

Prostor datových struktur

Většina běžně používaných struktur pro ukládání dat (gridy, hierarchie, sítě, …) obsahuje logické operace, které mohou uživatelé interaktivně na těchto strukturách provádět. Při implementaci je nutné učinit rozhodnutí o stupni automatizace použité operace, a zda budou interakce specifikovány přímo ve vizualizaci nebo v samostatném dialogovém okně. Při volbě automatizované techniky je nutné zvolit mezi důkladnými, ale časově náročnými technikami a rychlými, ale nepřesnými technikami. Nyní se pokusíme zaměřit na některé z těchto návrhových rozhodnutí.

Vezměme v úvahu uspořádání dimenzí pro vizualizaci multivariate dat. Plně manuální techniky mohou v tomto případě zahrnovat manipulaci s textovými vstupy v seznamu (pomocí operace posunu – nahoru a dolů, nebo pomocí techniky drag-and-drop). V případě paralelních souřadnic či matice bodových grafů můžeme manipulovat přímo s osami. U matice bodových grafů může posun jednoho prvku vyvolat změny v dalších řádcích a sloupcích, pokud chceme například zachovat symetrii podél hlavní diagonály.

Naopak automatické přeskládání dimenzí potřebuje alespoň dvě základní rozhodnutí o návrhu: jakým způsobem měřit kvalitu uspořádání a jakou strategii zvolit pro hledání těchto kvalitních uspořádání. Pro tato rozhodnutí je možné zvolit různé metriky. Jedna z běžně používaných je součet korelačních koeficientů mezi každou dvojicí dimenzí.

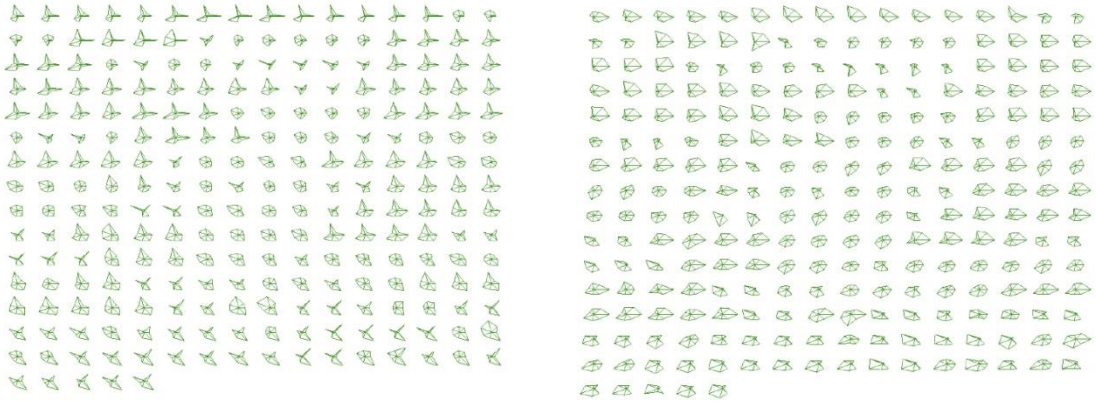
Tento **korelační koeficient** mezi dvěma dimenzemi je definován následovně:

$$\rho_{X,Y} = \frac{\sum (x_i y_i - n\mu_X \mu_Y)}{(n-1)\sigma_X \sigma_Y}$$

kde *n* je počet datových bodů, X a Y jsou dvě dimenze, x_i a y_i jsou hodnoty pro i-tý datový bod, μ_X je střední hodnota v X a σ_X je standardní odchylka pro X.

Další přístup k měření kvality uspořádání může zahrnout jednoduchost interpretace. Různá uspořádání dimenzí mohou vést k zobrazení s většími či menšími vizuálními shluky nebo strukturami. Například je udáváno, že při použití glyfů reprezentujících datové body je snazší analyzovat jednoduché tvary namísto komplexních. Proto pokud jsme schopni změřit

průměrnou nebo kumulativní složitost tvaru (např. počítáním prohlubní či vrcholů tvaru), můžeme tuto znalost využít pro porovnání vizuální složitosti různých uspořádání dimenzí.



Příklad je na obrázku, kdy levá část ukazuje původní uspořádání, zatímco pravá část zobrazuje výsledky po přeskládání dimenzí, kdy se snažíme redukovat konkávní oblasti glyfů a zvýšit procentuální podíl symetrických tvarů.

Pokud máme vybránu požadovanou kvalitu uspořádání, je dalším úkolem nalezení efektivní vyhledávací strategie pro nalezení těchto kvalitních uspořádání. Vyhodnocení všech možných uspořádání dimenzí je velmi náročné, pokud není počet dimenzí velmi malý (počet jednotlivých uspořádání je totiž N!). Tento počet může být dělen 2, pokud vezmeme v úvahu symetrická řešení, stále ale musíme vyhodnotit obrovský počet možností. Typickou strategií v těchto situacích je využití technik optimalizace. Problém uspořádání dimenzí je velmi podobný problému obchodního cestujícího, proto můžeme přímo použít algoritmy používané pro tento problém.

Jeden z nejjednodušších algoritmů pracuje následovně:

1. Vybereme dvě libovolné různé dimenze
2. Prohodíme jejich pozice a spočteme kvalitu uspořádání
3. Pokud je kvalita nižší než kvalita původního uspořádání, zrušíme prohození
4. Opakujeme kroky 1-3 fixně definovaným počtem iterací nebo dokud určitý počet provedených testů nevykazuje žádné zlepšení kvality

Tyto heuristické přístupy nejsou rozhodně optimální, nicméně často vedou k nalezení přijatelného řešení. Tyto přístupy se dají kombinovat i s manuálním přístupem, kdy uživatel může některá uspořádání zadat ručně na základě svých znalostí datové množiny a poté nechá systém automaticky dopočítat kvalitní uspořádání na jím modifikované množině.

Prostor vizualizace struktur

Některé z dříve popsaných technik je možné aplikovat i na prostor vizualizace struktur. Například technika rybího oka v prostoru obrazovky může být využita pro jakoukoliv vizualizaci struktury obsahující sekvence či gridy komponent. Stejná distorzní funkce může být v tomto případě použita pro nastavení rozestupů mezi osami paralelních souřadnic nebo pro nastavení velikosti buněk gridu při použití matice bodových grafů. Dalším příkladem je využití klastrování, filtrace či technik optimalizace (manuální i automatické) pro organizaci sad vizualizací na obrazovce jako jsou například objemové vizualizace obsahující stovky či tisíce různých sad s odlišným nastavením parametrů.

Klíčovým je ujištění, že uživatel je obeznámen se všemi operacemi, které může provádět nad strukturou vizualizace a využít konzistentní sadu ikon a označení, která je uživatel schopen rychle pochopit a používat. Další klíčovou funkcí je využití hladkých přechodů mezi vizualizacemi, což detailně probereme v následující sekci.

Animační transformace

V podstatě veškeré interakce v rámci vizualizačních systémů vedou ke změně zobrazeného obrázku. Některé z těchto změn jsou poměrně dramatické – například při otevření nové datové sady. Další změny mohou zachovat některé aspekty pohledu a jiné změnit. V případě, kdy si chce uživatel zachovat kontext zatímco svoji pozornost směřuje na měnící se oblast, je žádoucí poskytnout hladký přechod mezi výchozí a cílovou vizualizací. Například při rotaci s 3D objektem nebo datovou množinou je hladká změna orientace obecně mnohem lepší než skokový přechod do finální orientace. V některých případech k tomu stačí využít jednoduchou lineární interpolaci mezi počáteční a koncovou konfigurací. V jiných případech ale lineární interpolace nevede ke konstantní rychlosti změny (například při pohybu kamery po křivce). Navíc ve většině případů získáme mnohem přitažlivější výsledek za použití postupného zrychlení a zpomalení změny. V této sekci se tedy zaměříme na algoritmy, které musíme využít, pokud chceme dosáhnout této kontroly změn.

Prvním krokem algoritmů pro řízení změny v datech je získání uniformní parametrizace proměnné nebo proměnných, které chceme během animace řídit. Pro některé proměnné, jako například pozice podél rovné čáry nebo škálování, použijeme lineární interpolaci, která poskytne konzistentní změny v následujících časových krocích. Pro další proměnné, jako pozice podél zakřivené cesty, musíme problém přeformulovat zavedením nového parametru.

Předpokládejme, že původní parametr je funkce proměnné *t*, která nabývá hodnot od 0 do 1. Například můžeme použít kubický polynom pro spočtení (x, y) pozice pro různé hodnoty t: $x(t) = At^3 + Bt^2 + Ct + D$ (stejně pro y). Nyní můžeme vytvořit seznam pozic p_i pro $0 <= i <= n$,

kde *n* je počet kroků mezi počáteční a koncovou pozicí. Dělení *t* na *n* stejných podintervalů dosáhneme jednoduchým přiřazením příslušných hodnot *t* do parametrické rovnice.

Poté můžeme odhadnout délku oblouku A součtem vzdáleností mezi po sobě jdoucími body:

$$A = \sum_{i=1}^{i=n} dist(p_{i-1}, p_i)$$

Je zřejmé, že čím je menší krok mezi sousedními body, tím je přesnější odhad délky oblouku.

Pro většinu křivek je však vzdálenost mezi sousedními body různá. Proto kdybychom použili vždy výše popsaný přístup, byla by rychlost výsledné animace proměnlivá.

Při výpočtu délky oblouku je rovněž užitečné pro každý bod p_i spočítat vzdálenost d_i od počátku křivky k tomuto bodu.

Pak spočteme funkci A(i), která reprezentuje procentuální poměr vzdálenosti, kterou bod urazí v i-tém časovém kroku.

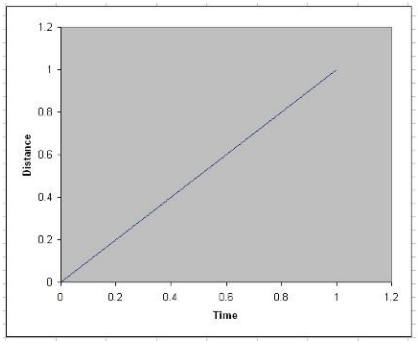
Pro zjednodušení použijeme místo proměnné i proměnnou *t* (0.0 <= t <= 1.0). Dále definujeme nový parametr *s* = A(t). Výsledky uložíme do tabulky, takže pro každou hodnotu *t* známe její odpovídající hodnotu *s* = A(t). Hodnotu *s* využijeme pro určení uniformní rychlosti (za využití lineární interpolace).

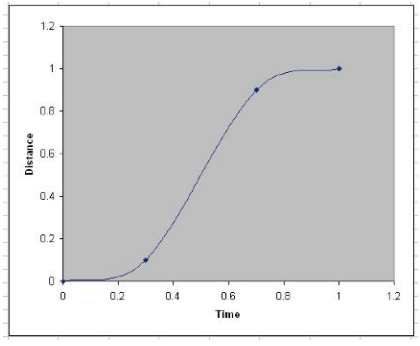
Výše uvedený postup je označován jako **reparametrizace**. Parametr *s* nyní slouží k ovládání rychlosti. Když vykreslíme parametr *s* v závislosti na čase, dostaneme rovnou čáru vedoucí z počátku (0.0, 0.0) do (1.0, 1.0). Jinými slovy, na začátku animace začínáme v původní pozici a když čas dosáhne hodnoty 1.0, jsme v koncové pozici. Rychlost jednoduše odpovídá sklonu této křivky. Co však s případy, kdy křivka není rovná, ale zakřivená? Pak části s malým sklonem představují nízkou rychlost a naopak velký sklon reprezentuje vysokou rychlost.

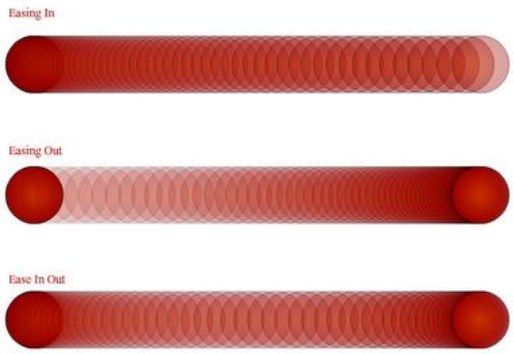
Protože počáteční a koncový bod jsou fixní, máme zajištěno, že skončíme, kde chceme.

Pro určení animace mezi počátečním a koncovým bodem máme nekonečně mnoho možností pro nastavení. Můžeme dokonce na určitý časový okamžik animaci zastavit. Hlavním předpokladem je, že křivka se monotónně zvyšuje a nyní předpokládejme rovněž, že se nemůže vracet zpět.

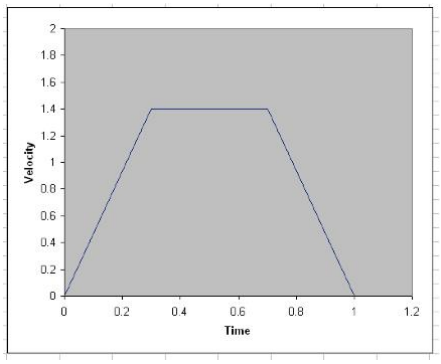
Běžným typem křivky pro řízení animace je křivka, která reprezentuje postupné navyšování rychlosti na začátku animace z nuly na požadovanou rychlost a poté opět postupné snižování rychlosti až k nule na konci animace. Toto chování přesně reprezentuje sinová křivka. Klíčovým požadavkem je udržení hladké křivky.





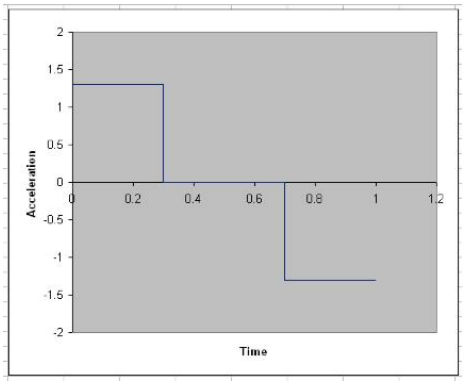


Někdy je jednodušší specifikovat pohyb pomocí **křivky rychlosti**. Rychlost je jednoduše první derivací křivky pozice. Křivka rychlosti pro případ postupného navyšování a snižování rychlosti se skládá ze segmentu, který se rovnoměrně zvyšuje od nuly, dále rovného segmentu a na konci z klesajícího segmentu končícího v nule (viz obrázek).



Prostor pod křivkou musí odpovídat hodnotě 1.0 – protože když je požadovaná rychlost příliš velká, musíme strávit více času při vzestupné a poté sestupné části.

Třetím typem křivky, která se občas používá pro kontrolu pohybu, je **akcelerační křivka**. Odpovídá druhé derivaci poziční křivky nebo analogicky první derivaci křivky rychlosti. Tvar křivky reprezentující postupně se zvyšující a poté snižující rychlost je složen ze tří horizontálních úsečkových segmentů – jeden nad osou (kladné zrychlení), jeden na ose (konstantní rychlost) a jeden pod osou (zpomalení).



Relativní pozice a délky čar nad a pod osou mohou být využity pro různé efekty a nemusí být nutně symetrické. Nicméně prostory definované vzestupnou a sestupnou fází se musí rovnat.

Poziční křivka a křivka rychlosti a akcelerace mohou být využity pro řízení jakéhokoliv atributu, který se v průběhu animace mění.

Následující pseudokód renderuje bodový graf kružnic, které jsou animovány v čase mezi dvěma množinami dimenzí x, y a r (poloměr) za použití lineární interpolace.

Argument *numFrames* specifikuje počet snímků animace a parametr *delay* určuje zpoždění mezi snímky v milisekundách.

```
Scatterplot-Animate(xDim1, yDim1, rDim1, xDim2, yDim2, rDim2, cDim, rMin, rMax,
numFrames, delay)
1       for each frame f from 0 to numFrames
2               do for each record i            //for each record,
3                       do x1 ← Normalize(i, xDim1)    // derive first state,
4                       y1 ← Normalize(i, yDim1)
5                       r1 ← Normalize(i, rDim1, rMin, rMax)
6                       x2 ← Normalize(i, xDim2)   // derive second state,
7                       y2 ← Normalize(i, yDim2)
8                       r2 ← Normalize(i, rDim2, rMin, rMax)
9                       p ← f/numFrames   // compute percent complete,
10                      x ← x1 + (x2 − x1) ∗ p   // and current state.
11                      y ← y1 + (y2 − y1) ∗ p
12                      r ← r1 + (r2 − r1) ∗ p
13                      MapColor(i, cDim)   // derive color, then
14                      Circle(x, y, r)   // draw the record as a circle.
15              Sleep(delay)   // Pause between frames.
```

Řízení interakce

V každé fázi interakční pipeline uživatel požaduje mechanismy pro řízení typu, umístění a stupně interakce při navigaci v prostoru dat i v samotné vizualizaci. Realizace tohoto řízení musí být intuitivní, jednoznačná a na úrovni detailu, která odpovídá prostoru, ve kterém se pohybujeme. Konkrétně se nyní zaměříme na typické řídicí prvky a jejich vhodné implementace. Tento seznam samozřejmě není vyčerpávající.

- **Výběr středu zájmu (Focus selection)**
  Výběr je nejčastěji uskutečňován pomocí nástrojů přímé manipulace, například za použití myši či jiného zařízení. V prostoru obrazovky a objektu toho dosáhneme klasickými operacemi výběru. V prostoru dat může být zapotřebí určit n-dimenzionální umístění. V závislosti na metodě zobrazení může toto znamenat násobné selekce. V prostoru atributů a struktur je nejdříve potřeba mít nejdříve grafické znázornění struktury nebo rozsahu atributů (například zobrazení stromu, tabulky nebo křivky ukazující rozsah barev v barevné mapě). Konečně fokus (ohnisko zájmu) může být definován i implicitně, pokud předpokládáme, že střed zájmu leží ve středu rozsahu interakce, který může být specifikován pomocí technik popsaných dále.
- **Výběr rozsahu (Extent selection)**
  Určení rozsahu interakce je obecně závislé na typu interakce a prostoru, ve kterém je interakce aplikována. Rozsah může být určen buď přímou manipulací nebo různými nástroji rozhraní. Může být definován jednou hodnotou (např. poloměr nebo maximální počet položek), vektorem hodnot (např. rozsah pro každou dimenzi nebo sada omezení). V mnoha systémech je rozsah často nastaven pevně.
- **Výběr typu interakce**
  Vzhledem k velkému počtu možných interakcí a různorodosti prostorů, ve kterých se používají, mělo by rozumné rozhraní obsahovat dvojici menu: jedno pro výběr prostoru a druhé pro specifikaci třídy interakce.
- **Výběr stupně interakce**
  Stupeň interakce je důležitým kontrolním parametrem, který může být specifikován pomocí jedné hodnoty (např. velikost škálování v oblasti středu zájmu). Je vhodné zpřístupnit slider pro ovlivňování tohoto parametru a tlačítko pro defaultní nastavení na minimální úroveň interakce.
- **Výběr typu míchání**
  Pokud je potřeba využít současně více různých interakčních technik, musíme nastavit strategii pro smíchání interakcí v regionech, které jsou ovlivněny více interakcemi. Toho se nejlépe dosáhne pomocí menu s různými možnostmi. Dostupné možnosti jsou závislé na prostoru, ve kterém se interakce projevuje a rovněž na typu použité interakce.

Algoritmy výběru

Následující pseudokód získává sadu vybraných záznamů z bodového grafu založeného na výběrovém obdélníku definovaném uživatelem.

```
Scatterplot-Select(xDim, yDim,xMin, xMax, yMin, yMax)
1       s ← ∅  // Initialize the set of records
2       for each record i        // For each record,
3          do x ← Normalize(i, xDim)  // derive the
                location,
4             y ← Normalize(i, yDim)
5              if xMin < x < xMax and yMin < y < yMax
6                 do s ← s ∪ i // select points within the
        rectangle.
7       return s
```

Bod v polygonu

Následující pseudokód určuje, zda je daný body uvnitř daného polygonu. Tento úkol se používá například pro určení, který polygon v choropletové mapě nebo glyf v bodovém grafu odpovídá dané pozici myši. Pseudokód využívá algorimus, který spočítá počet průsečíků paprsku vycházejícího z daného bodu s hranami polygonu. Pokud počet průsečíků paprsku s polygonem je lichý, pak je bod uvnitř polygonu.

```
Point-In-Polygon(xs, ys,numPoints, x, y)
1       j ← numPoints − 1
2       oddNodes ←false
3       for i ← 0 to numPoints − 1
4         do if ys[i] < y and ys[j] >= y or ys[j] < y and ys[i] >= y
5            do if xs[i] + (y − ys[i])/(ys[j] − ys[i]) ∗ (xs[j] − xs[i]) < x
6                 do oddNodes ← not oddNodes
7          j ← i
8       return oddNodes
```