

CS 277 - Experimental Haptics

Lecture 9 “Haptic Rendering *Tricks*”



Haptic “Tricks”: an Informal Definition

- Haptic rendering effects that may
 - Contribute to the realism of a VE
 - Cut your interface a break taking advantage of limitations in the human perceptual system

A list of tricks that we will cover

- Rendering stiff virtual objects
 - i.e. how to take advantage of humans' poor perception of position
- Rendering stiffer / softer deformable objects
 - i.e. how visual perception dominates haptics
- Rendering a dragon using a plane
 - i.e. taking advantage of asymmetries of the haptic sense
- Rendering smooth objects: force shading
 - i.e. reality is not a mesh
- Rendering frictional effects
 - i.e. rendering pure shapes doesn't really feel right
- Rendering textures
 - i.e. there is more to objects than just friction

A list of tricks that we will cover

- Rendering 3D shapes using 2 DOFs
 - i.e. how to project positions and forces on smaller rank vectorial spaces
- Rendering 2D shapes using 1 DOF
 - i.e. how work can be your ally (and your enemy)
- Rendering small bumps to feel **large**
 - i.e. how our sensitivity to force direction is not that good
- Rendering **large** virtual environments using small devices
 - i.e. how to take advantage of humans' poor perception of position
- Rendering fast cars without moving much
 - i.e. our vestibular sense is also pretty limited

A list of tricks that we will NOT cover

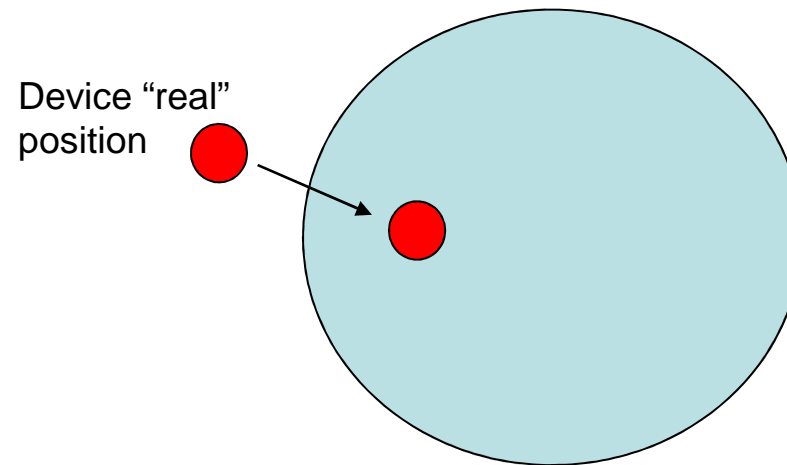
- But that should be fun
 - Vincent Hayward's paper that you will find online
 - i.e. tactile tricks
- Plenty more

A list of tricks that we will cover

- **Rendering stiff virtual objects**
 - i.e. how to take advantage of humans' poor perception of position
- Rendering stiffer / softer deformable objects
 - i.e. how visual perception dominates haptics
- Rendering a dragon using a plane
 - i.e. taking advantage of asymmetries of the haptic sense
- Rendering smooth objects: force shading
 - i.e. reality is not a mesh
- Rendering frictional effects
 - i.e. rendering pure shapes doesn't really feel right
- Rendering textures
 - i.e. there is more to objects than just friction

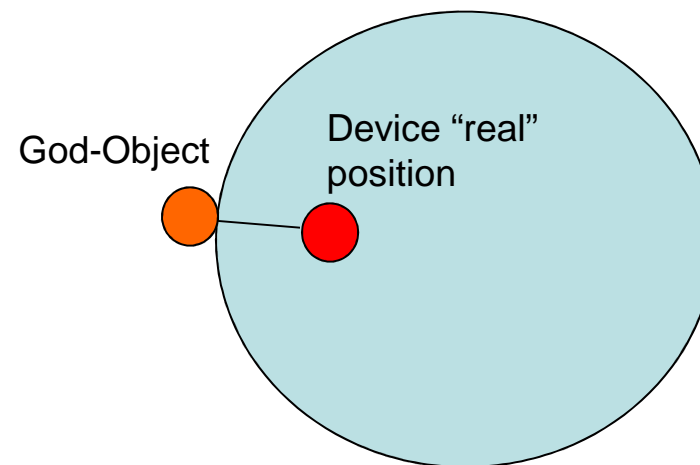
Rendering Stiff Virtual Objects

- This is an effect that you should be familiar with
- Basic idea:
 - Real device position will ALWAYS be inside of virtual object



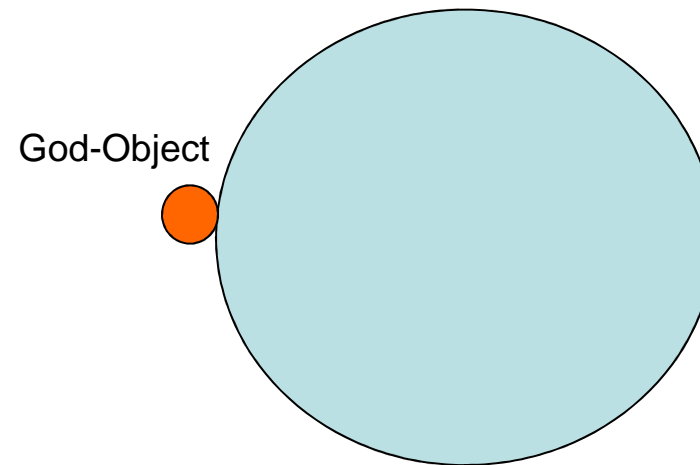
Rendering Stiff Virtual Objects

- This is an effect that you should be familiar with
- Basic idea:
 - Real device position will ALWAYS be inside of virtual object
 - All God-object like algorithms find a point on the surface
 - Such point can be used to compute forces
 - and as a visual representation of your finger



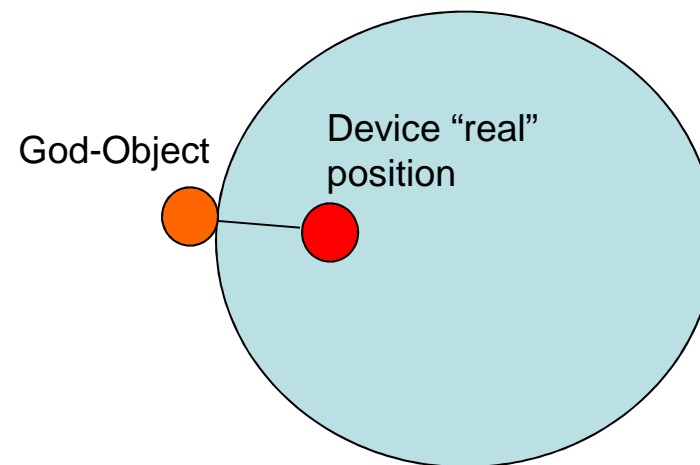
Rendering Stiff Virtual Objects

- This is an effect that you should be familiar with
- Basic idea:
 - Real device position will ALWAYS be inside of virtual object
 - All God-object like algorithms find a point on the surface
 - Such point can be used to compute forces
 - and as a visual representation of your finger
- Hiding “real” position visually aids the illusion of a stiff object



Rendering Stiff Virtual Objects

- Why does this work?
 - Visual feedback dominates our absolute position perception

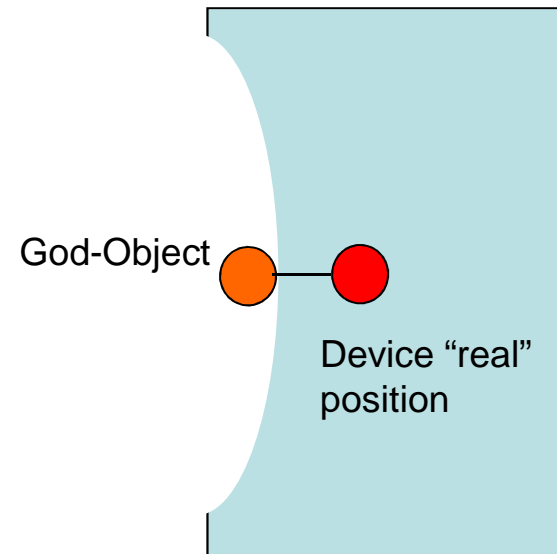


A list of tricks that we will cover

- Rendering stiff virtual objects
 - i.e. how to take advantage of humans' poor perception of position
- **Rendering stiffer / softer deformable objects**
 - **i.e. how visual perception dominates haptics**
- Rendering a dragon using a plane
 - i.e. taking advantage of asymmetries of the haptic sense
- Rendering smooth objects: force shading
 - i.e. reality is not a mesh
- Rendering frictional effects
 - i.e. rendering pure shapes doesn't really feel right
- Rendering textures
 - i.e. there is more to objects than just friction

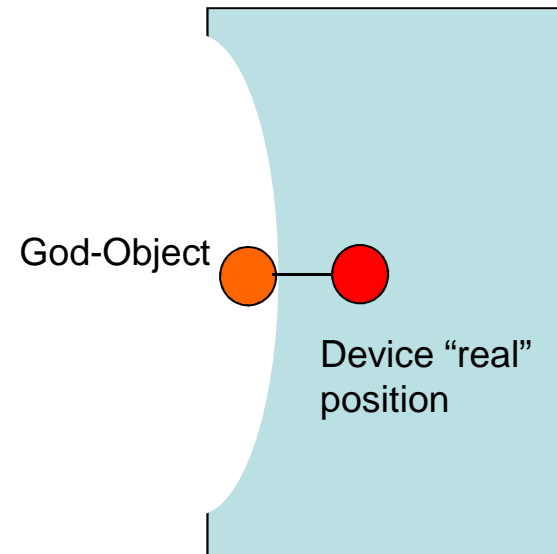
Rendering stiffer / softer deformable objects

- Similar to what we just discussed, but a step further
- Basic idea:
 - Deformable objects change shape when applying forces to them



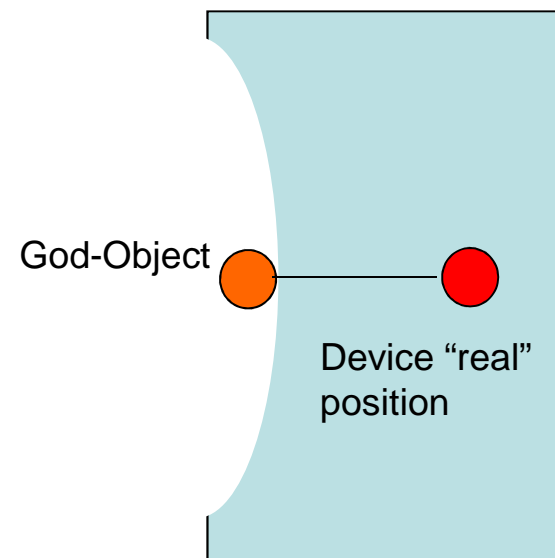
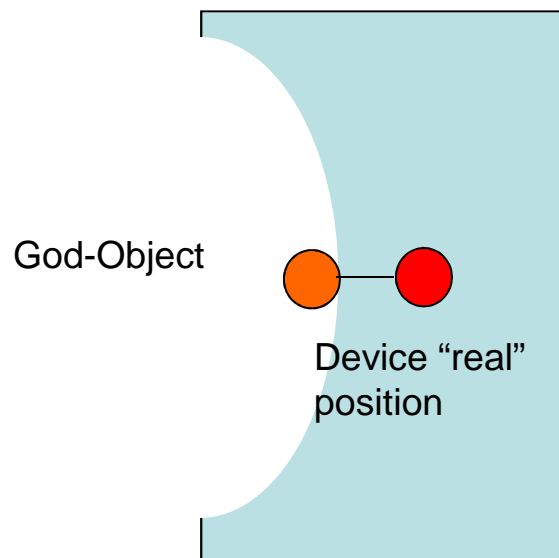
Rendering stiffer / softer deformable objects

- Similar to what we just discussed, but a step further
- Basic idea:
 - Deformable objects change shape when applying forces to them
 - The amount of deformation and real position of device do NOT have to match



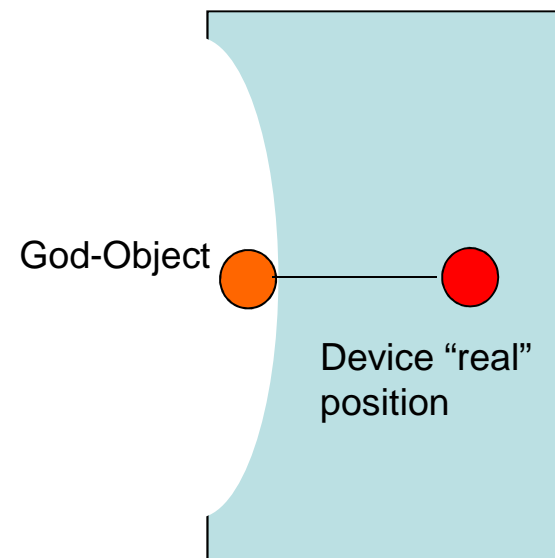
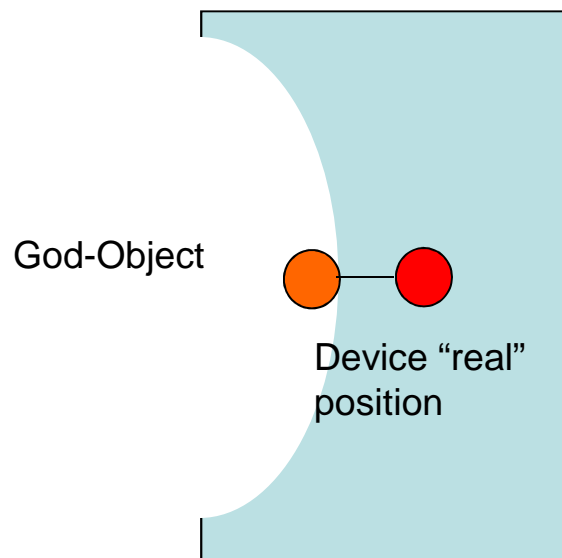
Rendering stiffer / softer deformable objects

- Similar to what we just discussed, but a step further
- Basic idea:
 - Deformable objects change shape when applying forces to them
 - The amount of deformation and real position of device do NOT have to match
- You can completely reverse the relationship between force and deformation



Rendering stiffer / softer deformable objects

- Why does this work?
 - There is a clear dominance over kinesthetic sense of hand position
 - And force channel capacity is limited

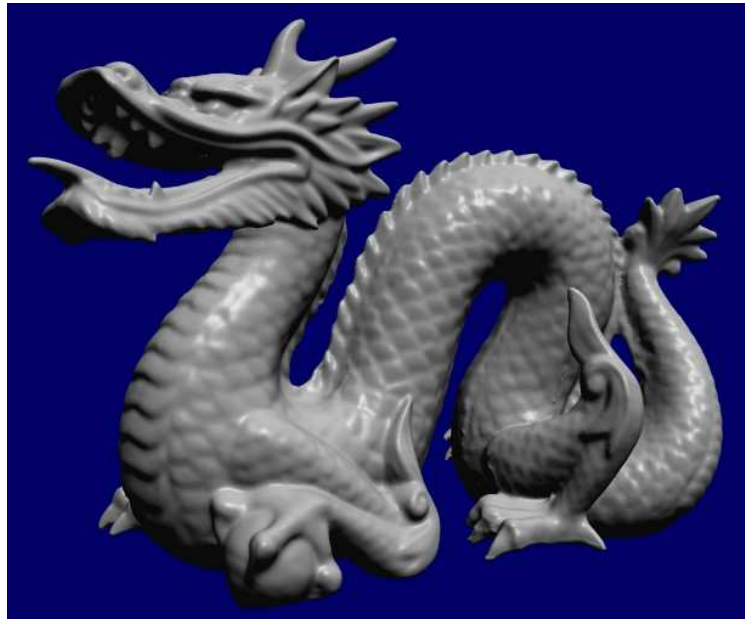


A list of tricks that we will cover

- Rendering stiff virtual objects
 - i.e. how to take advantage of humans' poor perception of position
- Rendering stiffer / softer deformable objects
 - i.e. how visual perception dominates haptics
- **Rendering a dragon using a plane**
 - **i.e. taking advantage of asymmetries of the haptic sense**
- Rendering smooth objects: force shading
 - i.e. reality is not a mesh
- Rendering frictional effects
 - i.e. rendering pure shapes doesn't really feel right
- Rendering textures
 - i.e. there is more to objects than just friction

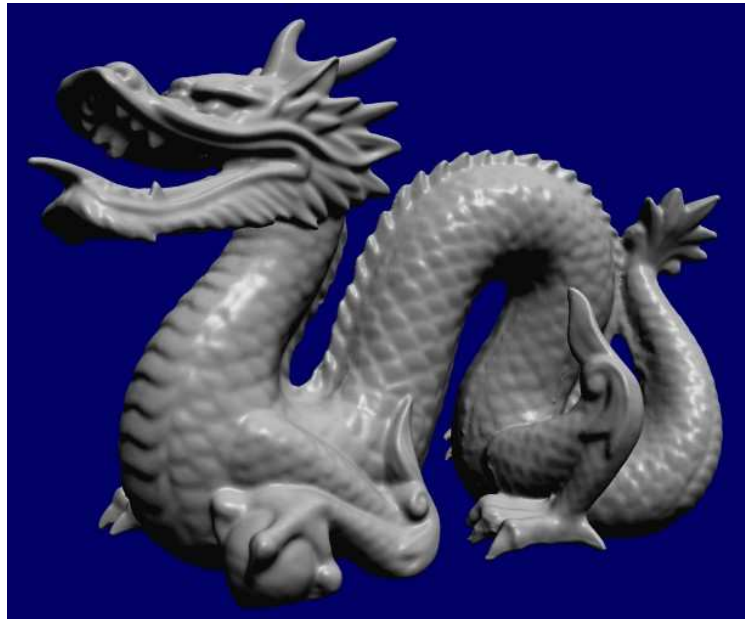
Rendering a Dragon Using a Plane

- Similar to what we discussed for implicit surfaces
- Basically:
 - Complicated surface may limit your haptic rendering rates



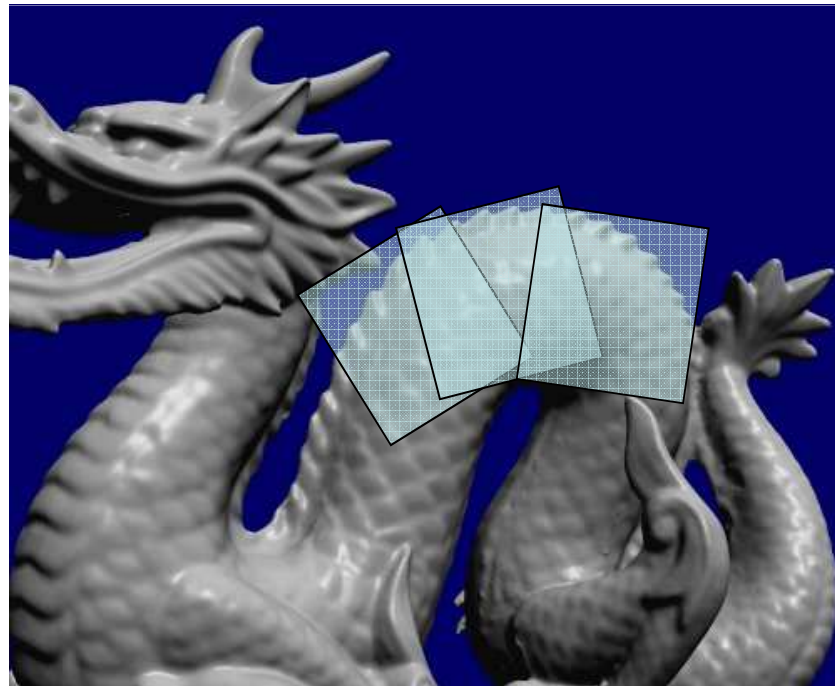
Rendering a Dragon Using a Plane

- Similar to what we discussed for implicit surfaces
- Basically:
 - Complicated surface may limit your haptic rendering rates
 - Decoupling collision detection and haptic rendering can help rendering stiffer objects



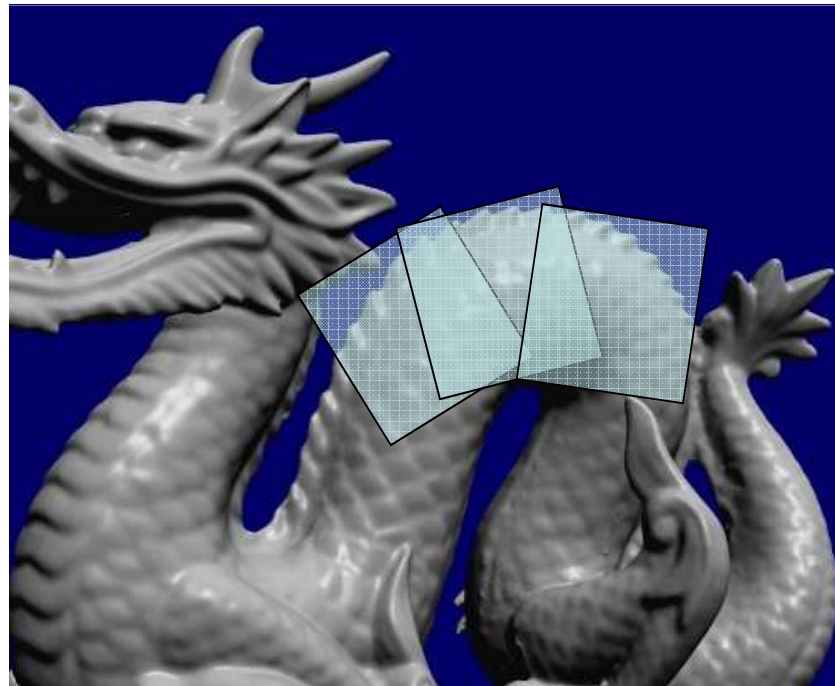
Rendering a Dragon Using a Plane

- In other words
 - Slow thread computes a new “local model” that approximates object surface but is simple (e.g plane, sphere, ...)
 - Fast thread computes fast collision detection and force rendering with local model



Rendering a Dragon Using a Plane

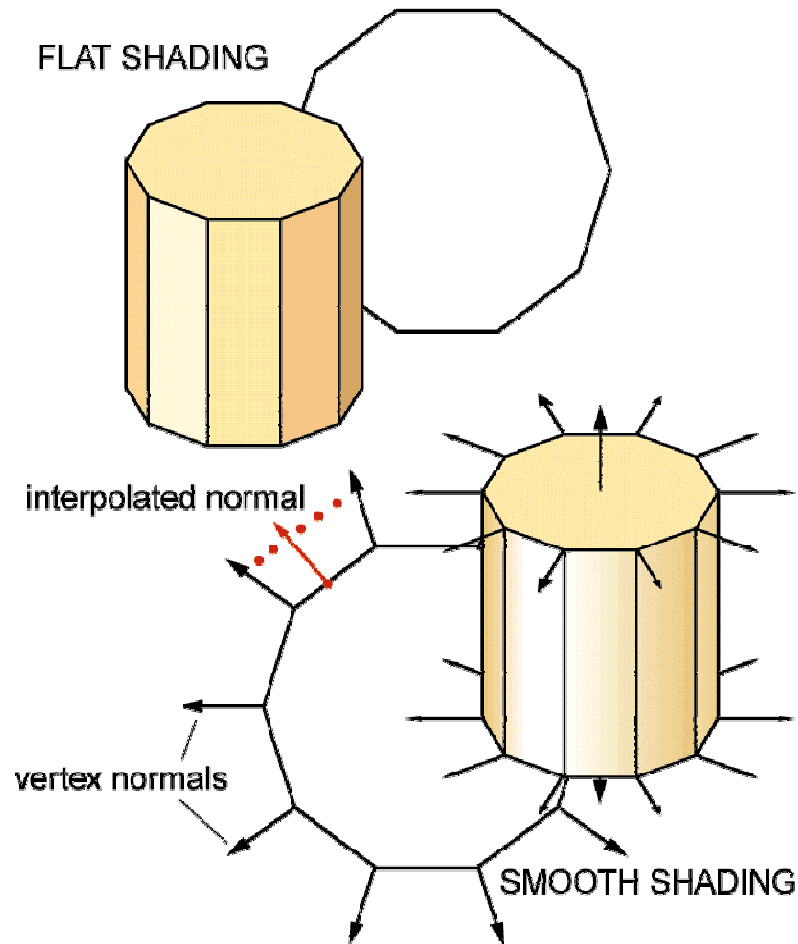
- Why does this work?
 - Max hand bandwidth is about 5Hz in motion => local model computation can be slow
 - But we perceive up to KHz => collision detection and force response needs to be as fast as possible



A list of tricks that we will cover

- Rendering stiff virtual objects
 - i.e. how to take advantage of humans' poor perception of position
- Rendering stiffer / softer deformable objects
 - i.e. how visual perception dominates haptics
- Rendering a dragon using a plane
 - i.e. taking advantage of asymmetries of the haptic sense
- **Rendering smooth objects: force shading**
 - **i.e. reality is not a mesh**
- Rendering frictional effects
 - i.e. rendering pure shapes doesn't really feel right
- Rendering textures
 - i.e. there is more to objects than just friction

Rendering Smooth Objects: Force Shading



Graphic Shading

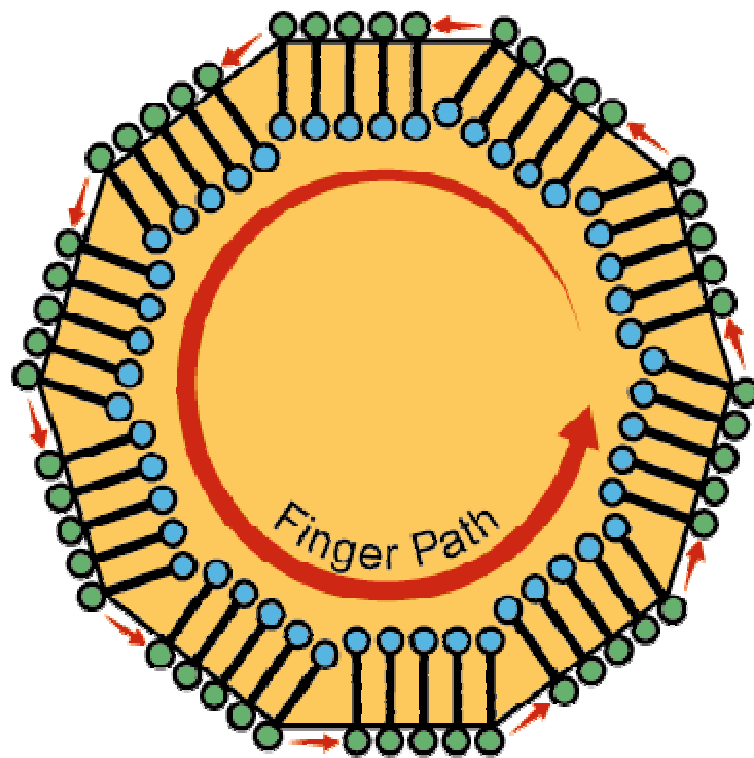
eliminate color
discontinuities

Haptic Shading

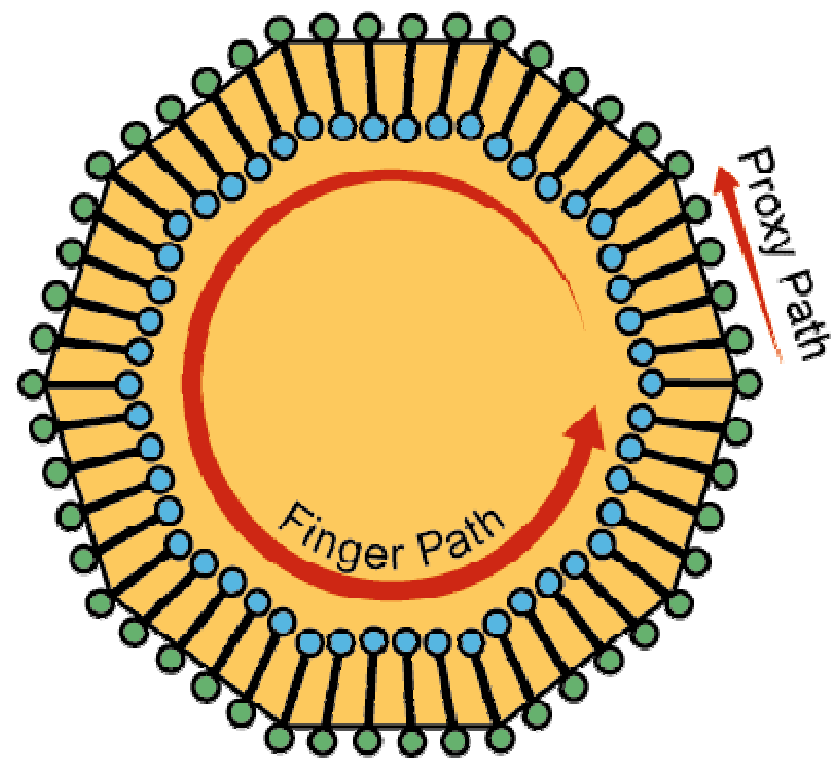
eliminate force
discontinuities

Rendering Smooth Objects: Force Shading

- Interpolate vertex normals across polygon to get continuous, smooth normals (just like Phong shading in graphics)



Faceted Cylinder



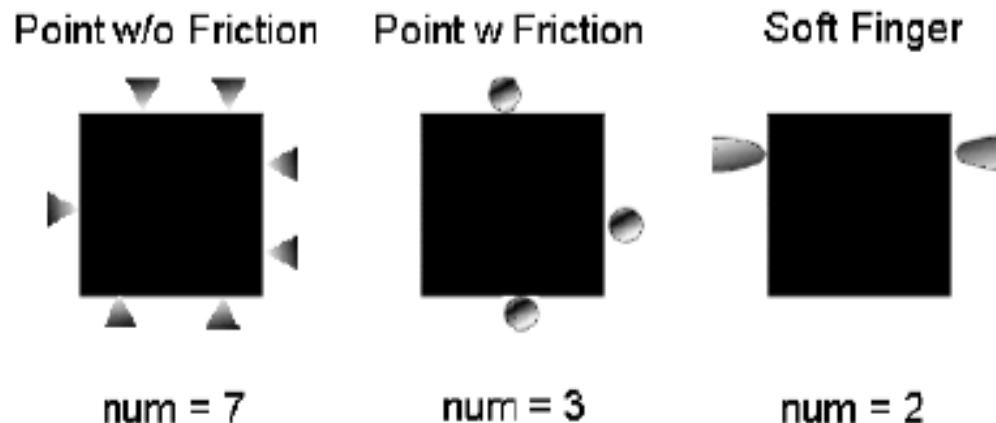
Shaded Cylinder

A list of tricks that we will cover

- Rendering stiff virtual objects
 - i.e. how to take advantage of humans' poor perception of position
- Rendering stiffer / softer deformable objects
 - i.e. how visual perception dominates haptics
- Rendering a dragon using a plane
 - i.e. taking advantage of asymmetries of the haptic sense
- Rendering smooth objects: force shading
 - i.e. reality is not a mesh
- **Rendering frictional effects**
 - **i.e. rendering pure shapes doesn't really feel right**
- Rendering textures
 - i.e. there is more to objects than just friction

Rendering Frictional Effects

- In reality mostly everything has friction
- Objects with friction
 - are perceptually more interesting
 - can be grasped and manipulated
- A frictionless sphere cannot be completely restrained




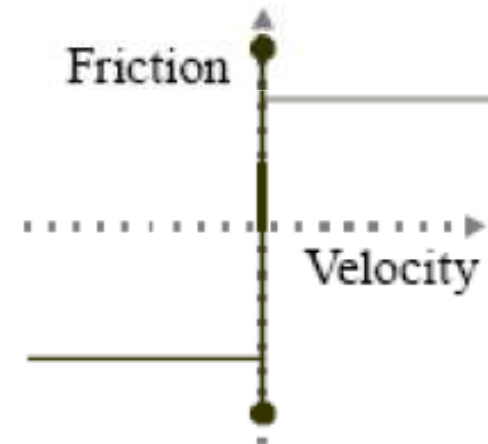
Rendering Frictional Effects

- Friction is the force resisting the relative motion of solid surfaces, fluid layers, or material elements sliding against each other.

Rendering Frictional Effects

- There are MANY models for friction
- We will deal a simple 2 state model:
 - Static Friction
 - (Coulomb) Dynamic friction

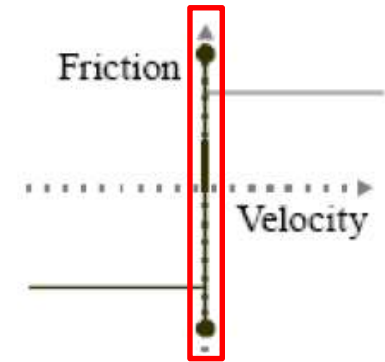
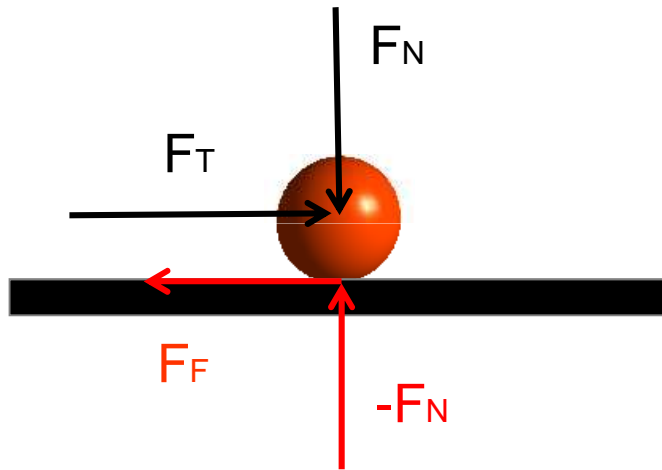

velocity independent



Rendering Frictional Effects

Static Friction

- not enough tangential force to “break away”
- 2 objects stick together



Static friction:

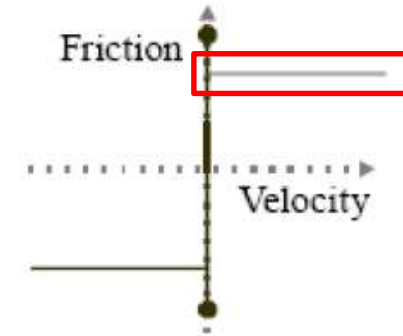
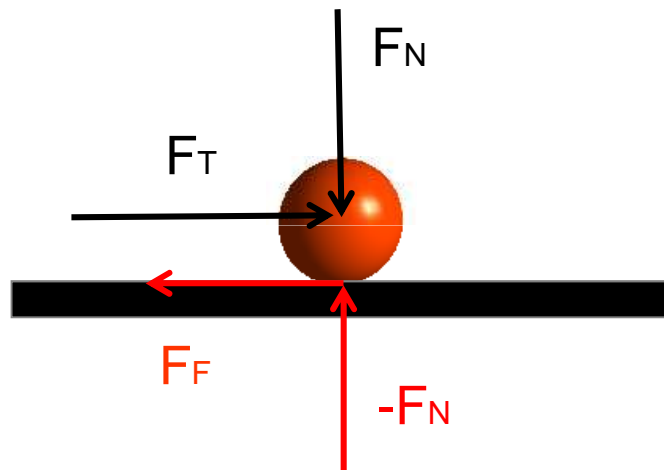
$$F_F \leq \mu_S F_N$$

If F_N is high enough NO motion takes place

Rendering Frictional Effects

Dynamic Friction

- enough tangential force to break away
- 2 objects slip
- smaller force opposes motion



Dynamic friction:

$$F_F > \mu_S F_N$$

$$F_T = \mu_D F_N$$

If F_F is high enough

- Motion happens
- A velocity independent force resists motion

Rendering Frictional Effects

- Note:

$$\mu_S > \mu_D$$

Rendering Frictional Effects

- Ok, practically speaking, how do I code this?
 - First you need to figure out which state you're in

$$F_F \leq \mu_S F_N$$

- If you are in static friction state
 - No motion
- If you are in dynamic friction state
 - Yes motion
 - $F_T = \mu_D F_N$

Rendering Frictional Effects

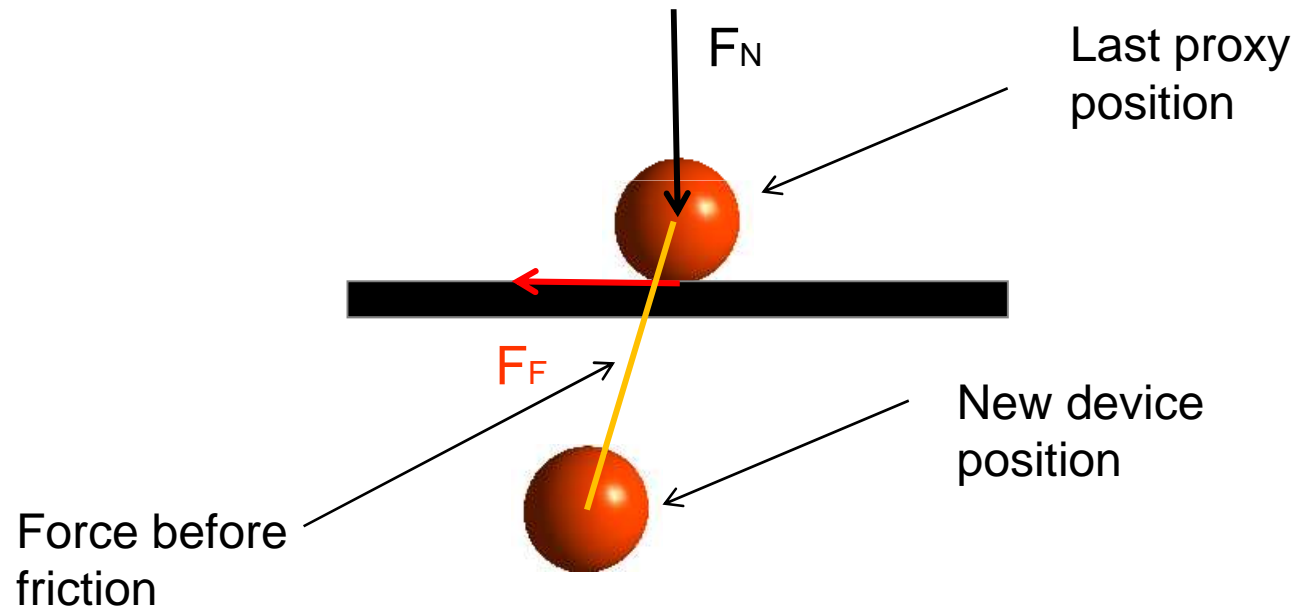
- Ok, practically speaking, how do I code this?
 - First you need to figure out which state you're in

$$F_F \leq \mu_S F_N$$

- If you are in static friction state
 - No motion
- If you are in dynamic friction state
 - Yes motion
 - $F_T = \mu_D F_N$

Rendering Frictional Effects

- What are F_F and F_N ?



Rendering Frictional Effects

- Ok, practically speaking, how do I code this?
 - First you need to figure out which state you're in

$$F_F \leq \mu_S F_N$$

- If you are in static friction state

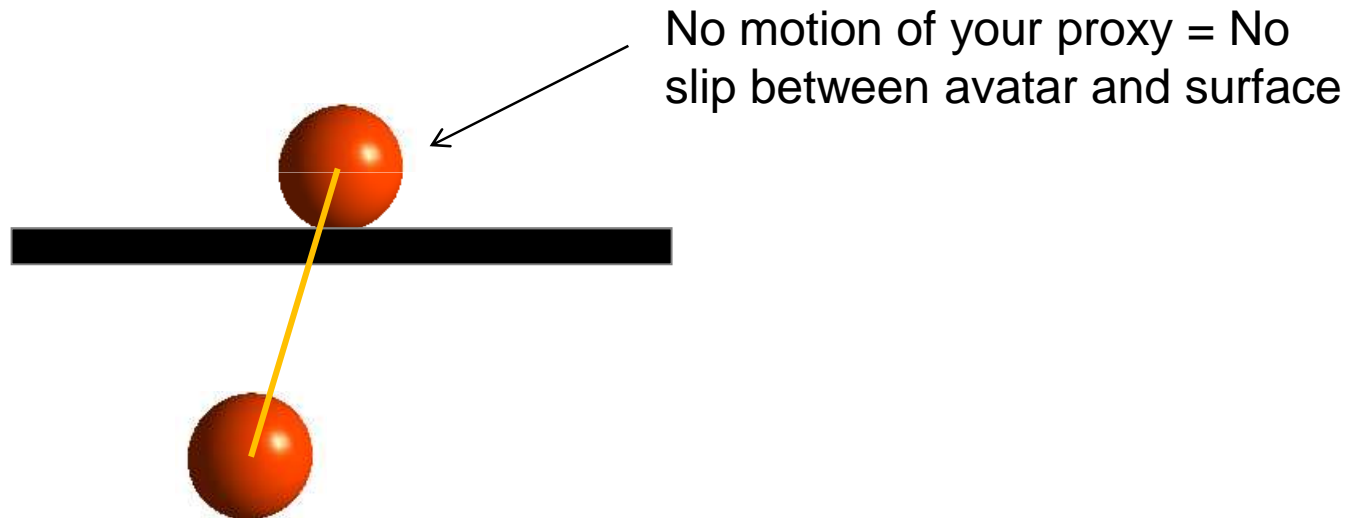
- No motion

- If you are in dynamic friction state

- Yes motion
- $F_T = \mu_D F_N$

Rendering Frictional Effects

- Even more practically:
 - How do I render No Motion?



Rendering Frictional Effects

- Ok, practically speaking, how do I code this?
 - First you need to figure out which state you're in

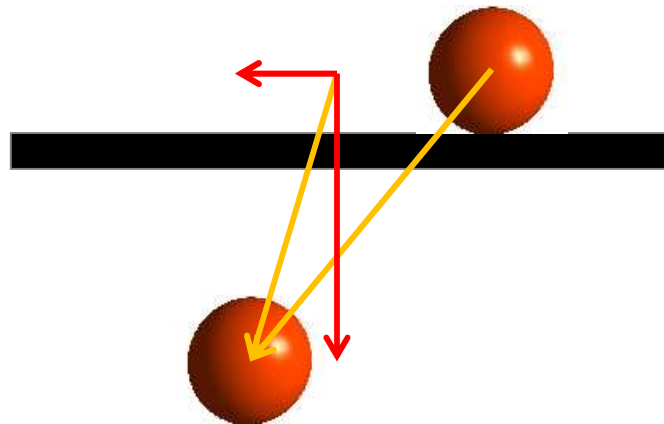
$$F_F \leq \mu_S F_N$$

- If you are in static friction state
 - No motion
- If you are in dynamic friction state

- Yes motion
- $F_T = \mu_D F_N$

Rendering Frictional Effects

- Even more practically:
 - How do I render $F_T = \mu_D F_N$ on user?
 - You know F_N and μ_D and thus F_T
 - You want the user to feel a tangential force = F_T
 - Don't let the proxy catch up to the actual device position



A list of tricks that we will cover

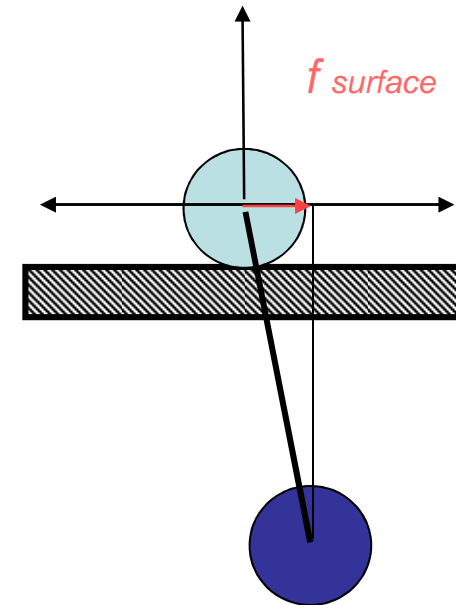
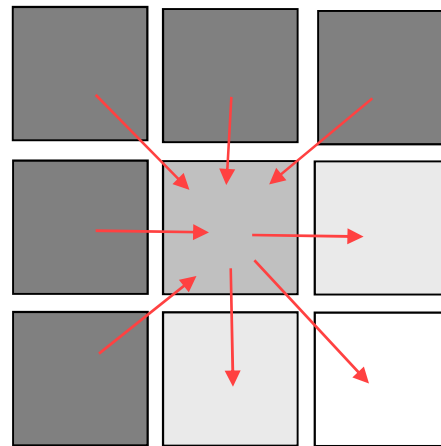
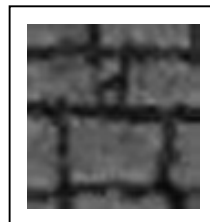
- Rendering stiff virtual objects
 - i.e. how to take advantage of humans' poor perception of position
- Rendering stiffer / softer deformable objects
 - i.e. how visual perception dominates haptics
- Rendering a dragon using a plane
 - i.e. taking advantage of asymmetries of the haptic sense
- Rendering smooth objects: force shading
 - i.e. reality is not a mesh
- Rendering frictional effects
 - i.e. rendering pure shapes doesn't really feel right
- **Rendering textures**
 - **i.e. there is more to objects than just friction**

Rendering Textures

32 bits / RGBA



8 bits / gray scale



Build local force gradient

Define force thresholds proportional to slopes

A list of tricks that we will cover

- **Rendering 3D shapes using 2 DOFs**
 - i.e. how to project positions and forces on smaller rank vectorial spaces
- Rendering 2D shapes using 1 DOF
 - i.e. how work can be your ally (and your enemy)
- Rendering small bumps to feel **large**
 - i.e. how our sensitivity to force direction is not that good
- Rendering **large** virtual environments using small devices
 - i.e. how to take advantage of humans' poor perception of position
- Rendering fast cars without moving much
 - i.e. our vestibular sense is also pretty limited

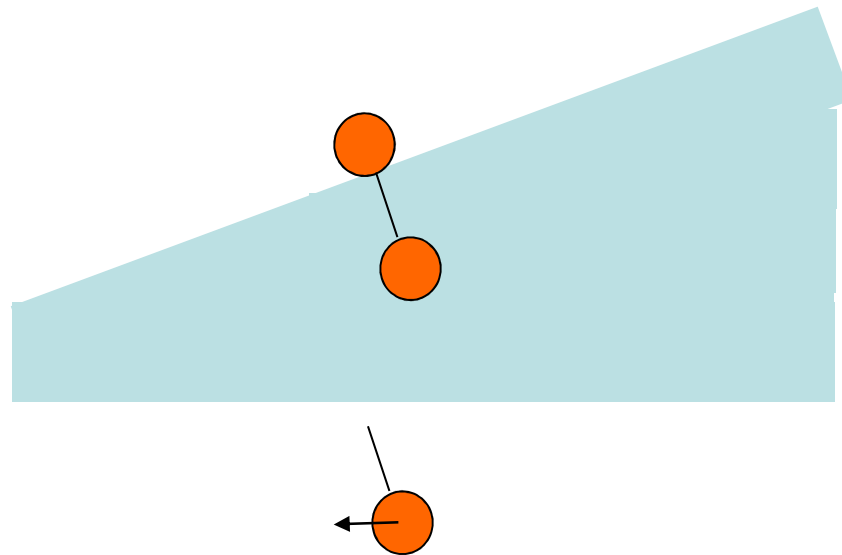
Rendering 3D shapes using 2 DOFs

- Why would you want to do this?
 - Assume a 2 sensor 2 actuator device
 - Try to render a 3D world through it



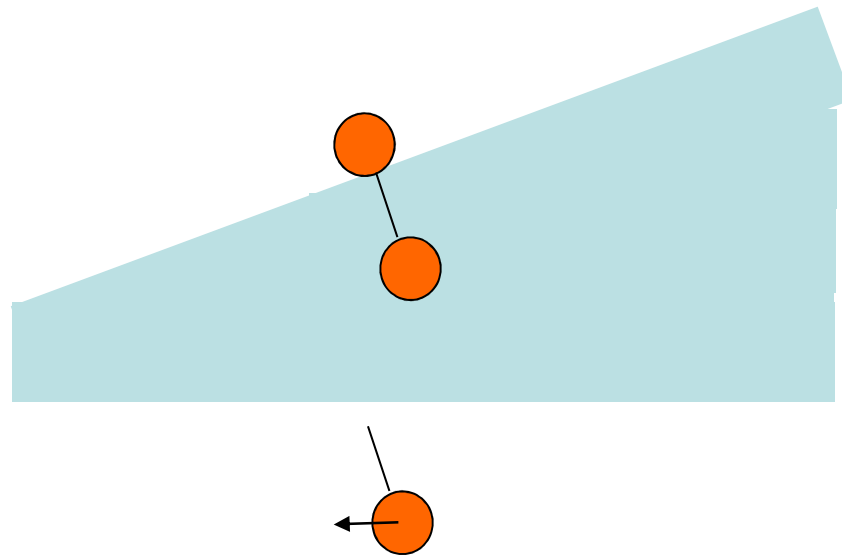
Rendering 3D shapes using 2 DOFs

- Very similar to what we just described for textures
- Basically, assuming interaction with surface $z = g(x, y)$
 - 3DOF device: $F = -r(p) n$
 - 2DOF device: $F_{\text{lateral}} = -r(p) \text{grad}(g)$



Rendering 3D shapes using 2 DOFs

- Assuming constant penetration > 0
 - you can feel a lateral force proportional to the steepness of your plane
 - as you move your device across the surface
- In other words, we're projecting the force on the display of the device

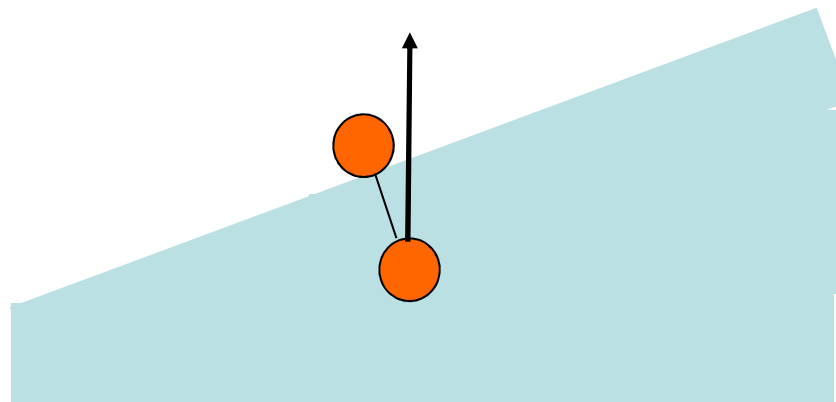


A list of tricks that we will cover

- Rendering 3D shapes using 2 DOFs
 - i.e. how to project positions and forces on smaller rank vectorial spaces
- **Rendering 2D shapes using 1 DOF**
 - **i.e. how work can be your ally (and your enemy)**
- Rendering small bumps to feel **large**
 - i.e. how our sensitivity to force direction is not that good
- Rendering **large** virtual environments using small devices
 - i.e. how to take advantage of humans' poor perception of position
- Rendering fast cars without moving much
 - i.e. our vestibular sense is also pretty limited

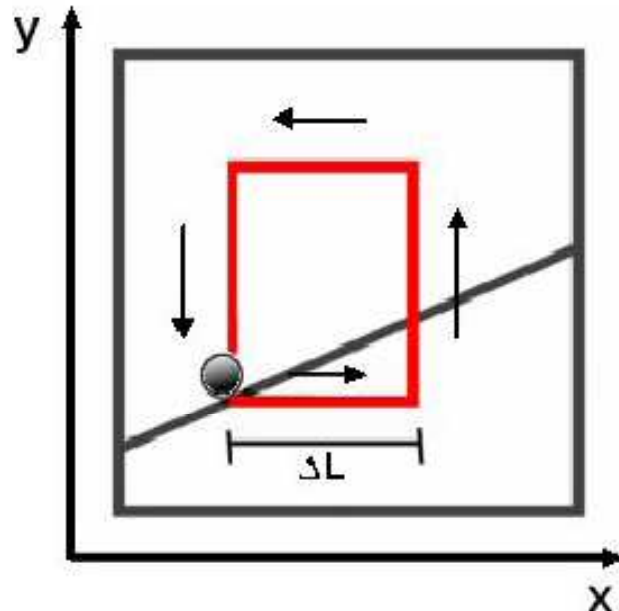
Rendering 2D shapes using 1 DOF

- Why would you want to do this?
 - Assume a 2 sensor 1 actuator device (asymmetric device)
 - Try to render a 3D world through it



Rendering 2D shapes using 1 DOF

- When moving to the left you load a spring for free
- Good news: this extra work gives you the impression of touching a 2D object
- Bad news: surface will feel extra active

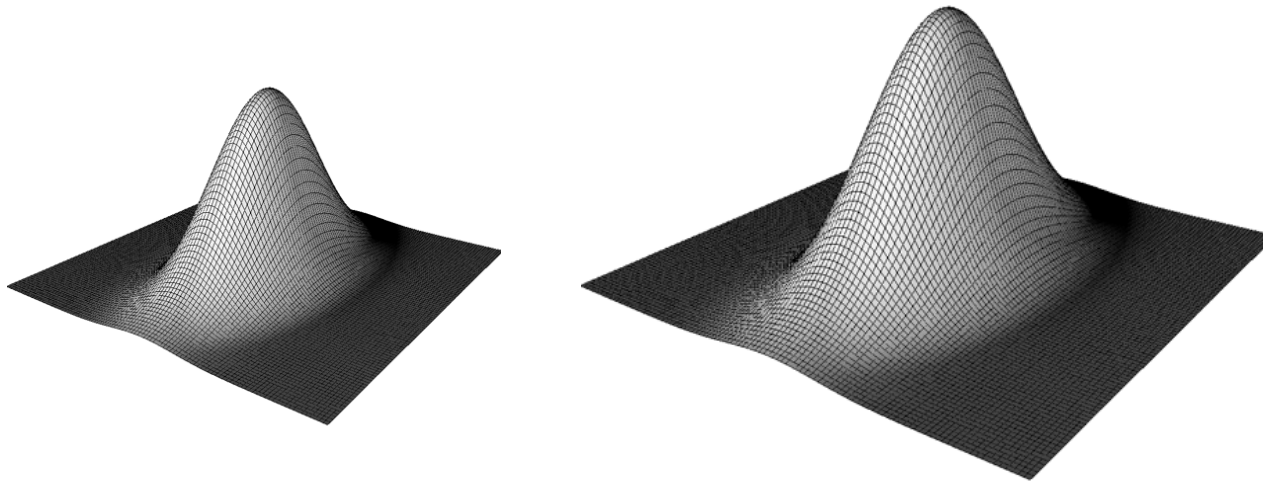


A list of tricks that we will cover

- Rendering 3D shapes using 2 DOFs
 - i.e. how to project positions and forces on smaller rank vectorial spaces
- Rendering 2D shapes using 1 DOF
 - i.e. how work can be your ally (and your enemy)
- **Rendering small bumps to feel large**
 - **i.e. how our sensitivity to force direction is not that good**
- Rendering **large** virtual environments using small devices
 - i.e. how to take advantage of humans' poor perception of position
- Rendering fast cars without moving much
 - i.e. our vestibular sense is also pretty limited

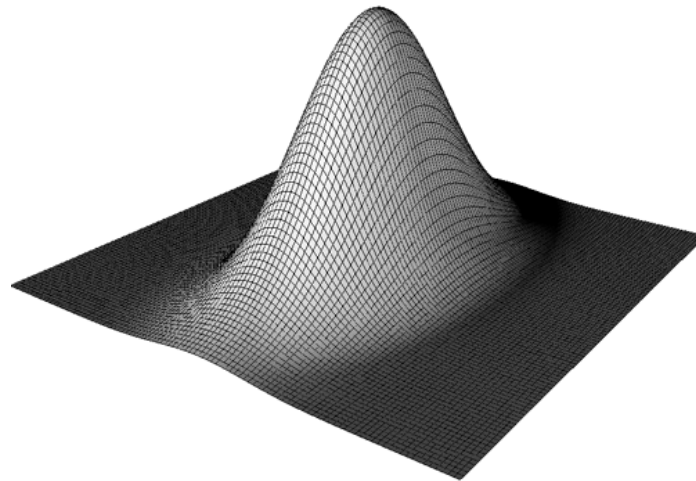
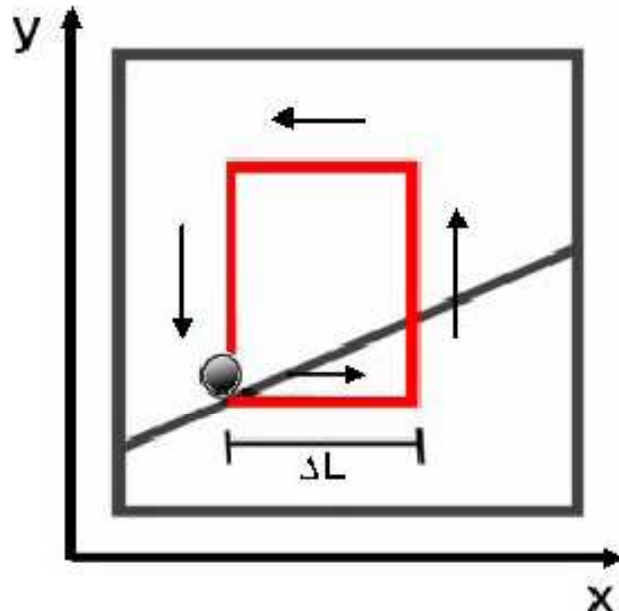
Rendering small Bumps to Feel **Large**

- As prof Tan explained, our force direction perception
 - Is poor
 - Can be greatly influenced by visual feedback
- You can take advantage of this and make smaller bumps feel larger by “cheating” visually



Rendering small Bumps to Feel **Large**

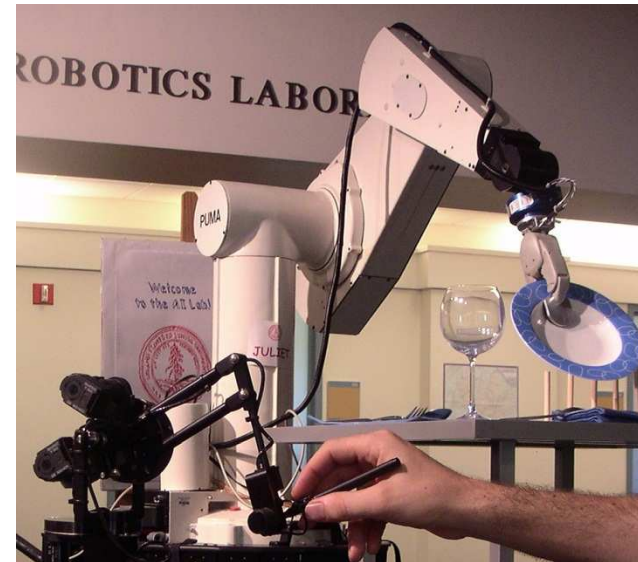
- This can be really useful if you want to render virtual objects using an asymmetric device



A list of tricks that we will cover

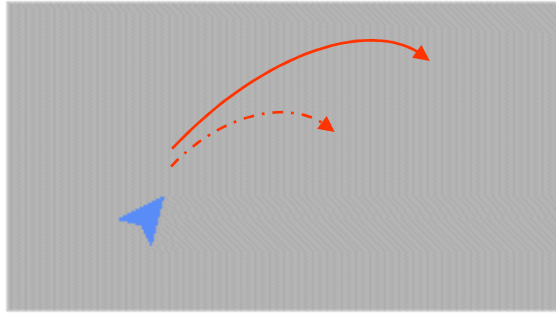
- Rendering 3D shapes using 2 DOFs
 - i.e. how to project positions and forces on smaller rank vectorial spaces
- Rendering 2D shapes using 1 DOF
 - i.e. how work can be your ally (and your enemy)
- Rendering small bumps to feel **large**
 - i.e. how our sensitivity to force direction is not that good
- **Rendering large virtual environments using small devices**
 - **i.e. how to take advantage of humans' poor perception of position**
- Rendering fast cars without moving much
 - i.e. our vestibular sense is also pretty limited

Exploring Large VEs Using small devices

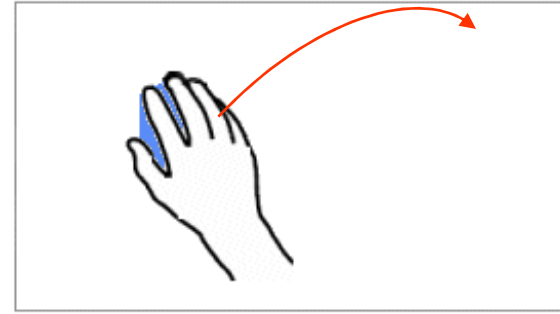


Control Paradigms

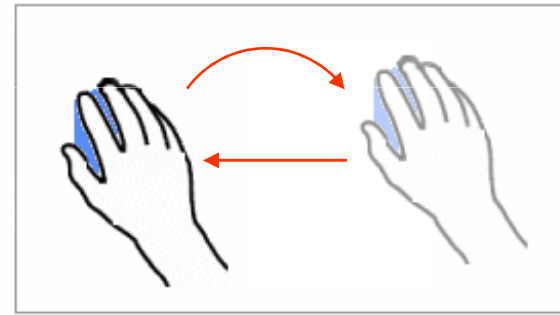
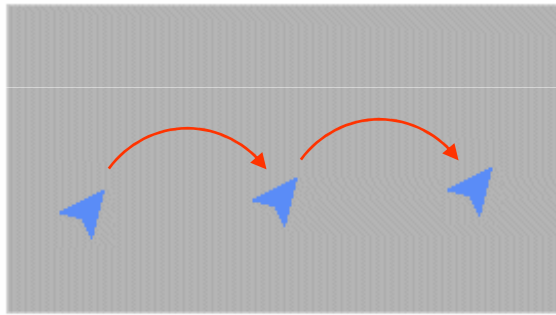
POSITION
SCALING



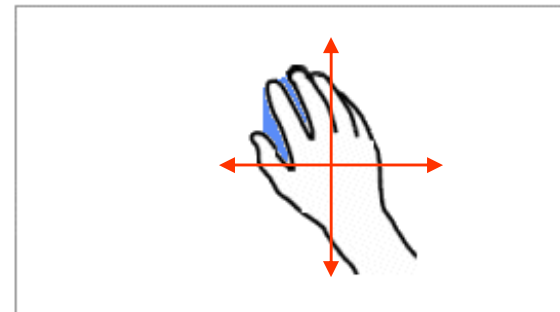
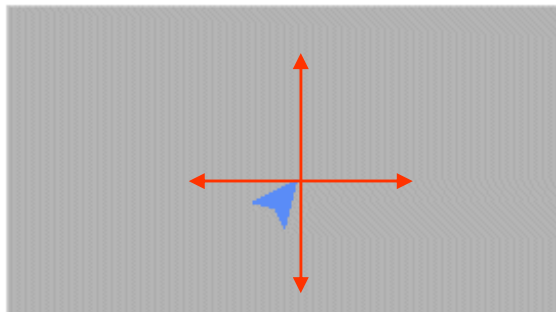
N:1



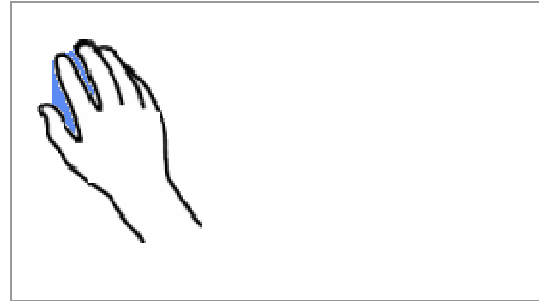
POSITION
INDEXING



RATE
CONTROL



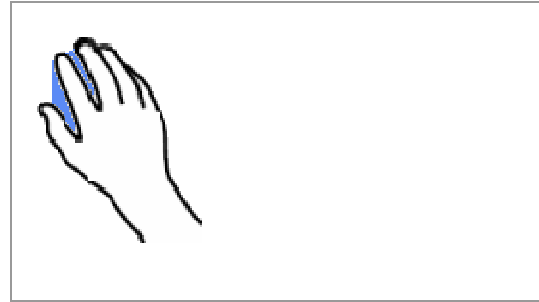
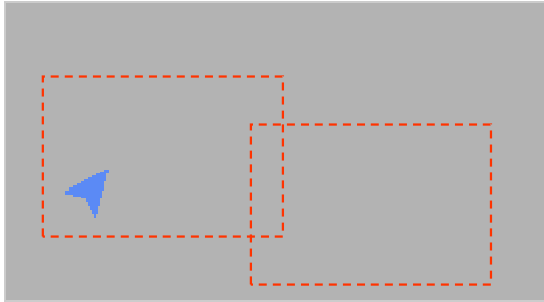
Position Control and Scaling



$$\vec{r}_a = k_s \cdot \vec{r}_d$$

- Position control is one of the most common control paradigms used today with computer mice or haptic interfaces.
- A bijective mapping between the physical and virtual workspace.
- Loss of spatial resolution when a high scaling factor k_s is used.
- Interacting with small objects becomes difficult if k_s is large.

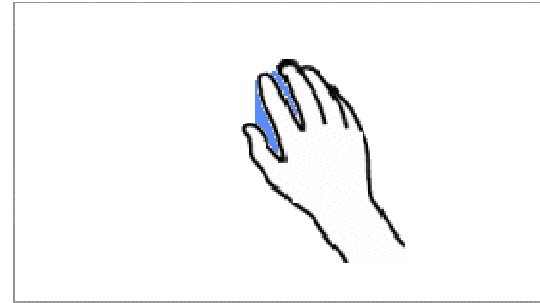
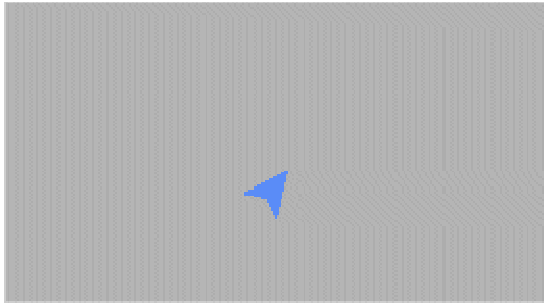
Indexing



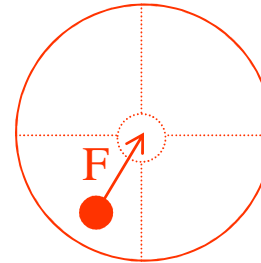
$$\vec{r}_a = k_s \cdot \vec{r}_d + \vec{r}_w$$

- The physical workspace of the device mapped inside the virtual environment is shifted **manually**.
- Indexing requires an additional input channel (switch) on ground based haptic devices.
- Indexing is not optimal for very small devices

Rate Control



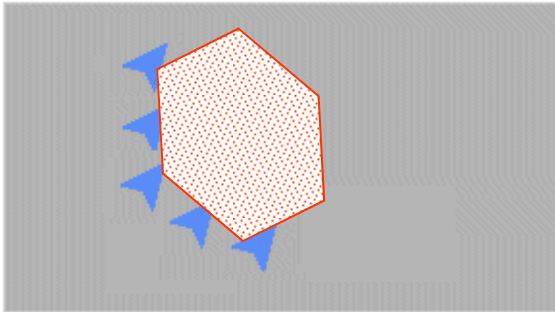
$$\vec{v}_a = k_r \cdot (\vec{r}_d - \vec{r}_{d0})$$



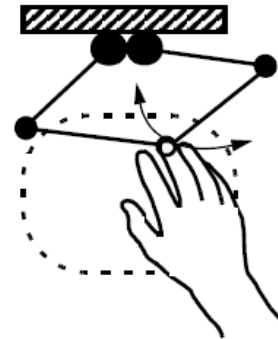
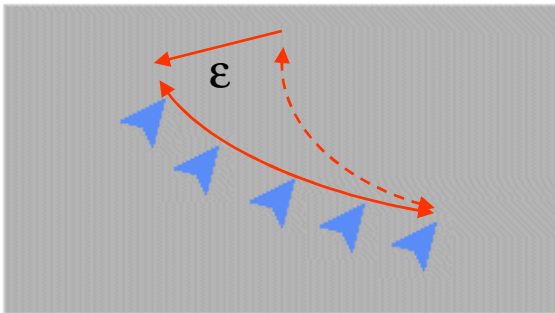
- Position of the hand dictates velocity of the cursor.
- Fast changes of directions of the cursor are not possible.
- Rate control enables the user to explore or tele-operate a robot in a very large workspace.

Observations

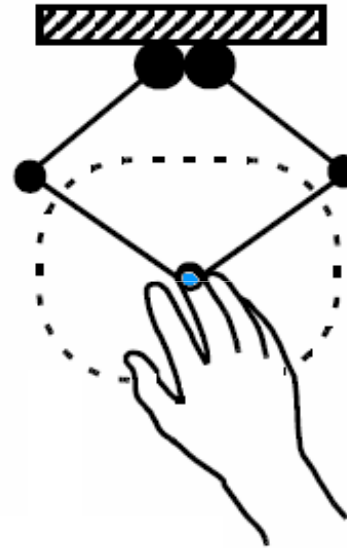
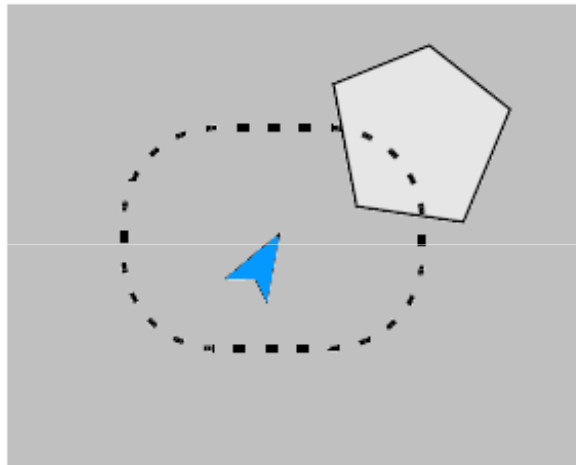
- interaction between a cursor and an object remains local.



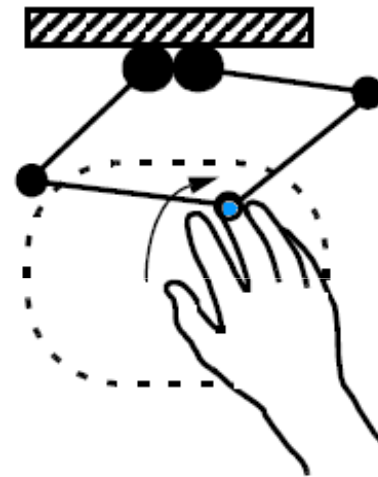
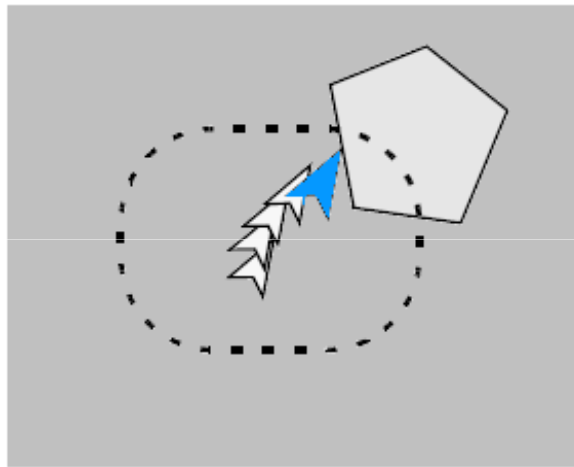
- People are greatly influenced by what they perceive visually and often do not notice small deviations of their hands.



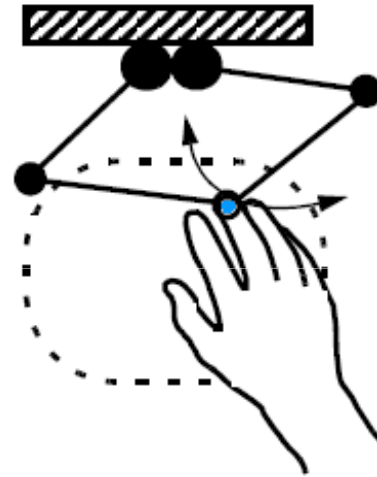
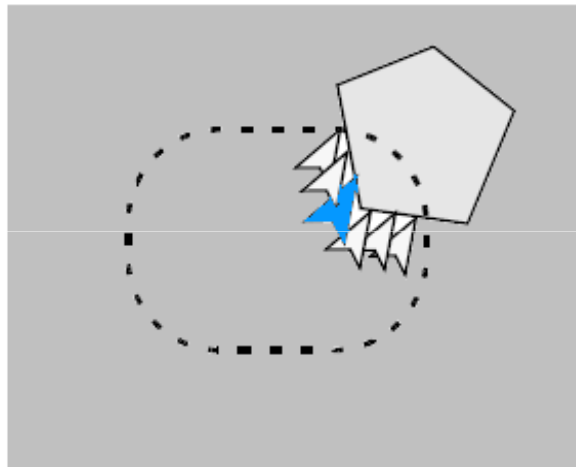
Workspace Drift Control



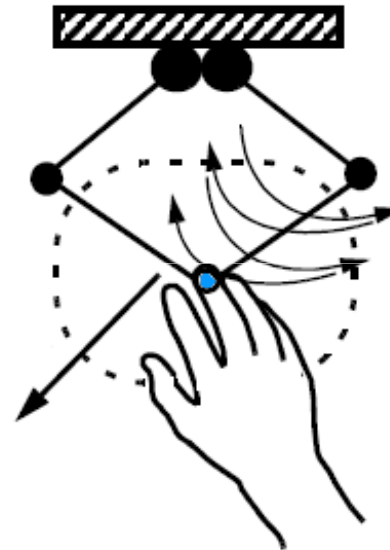
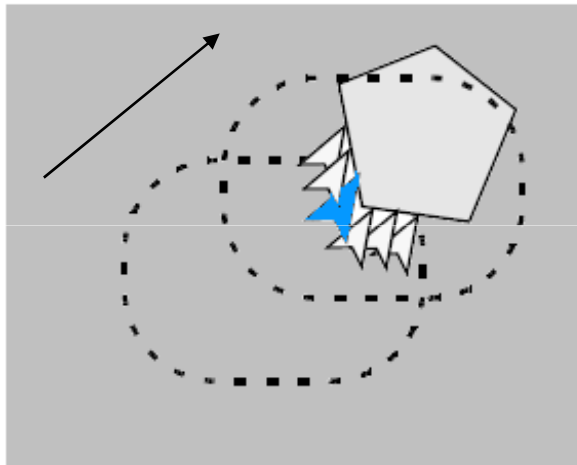
Workspace Drift Control



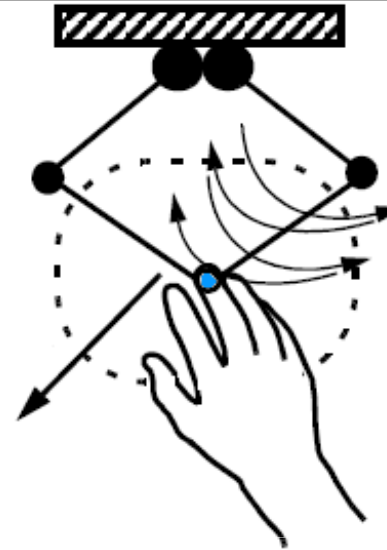
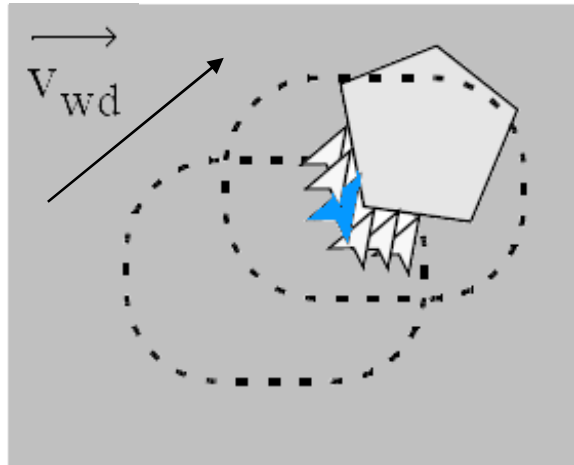
Workspace Drift Control



Workspace Drift Control



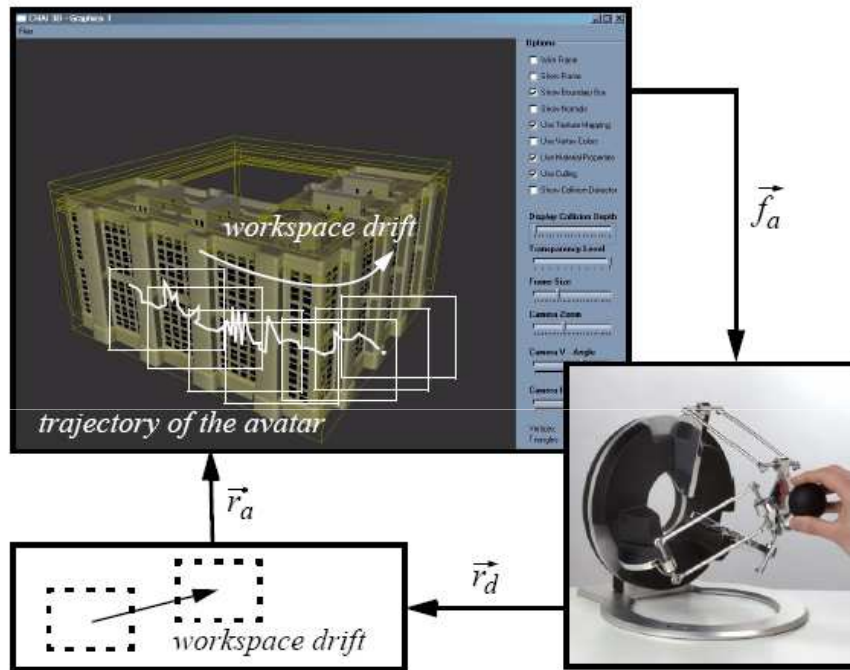
Workspace Drift Control



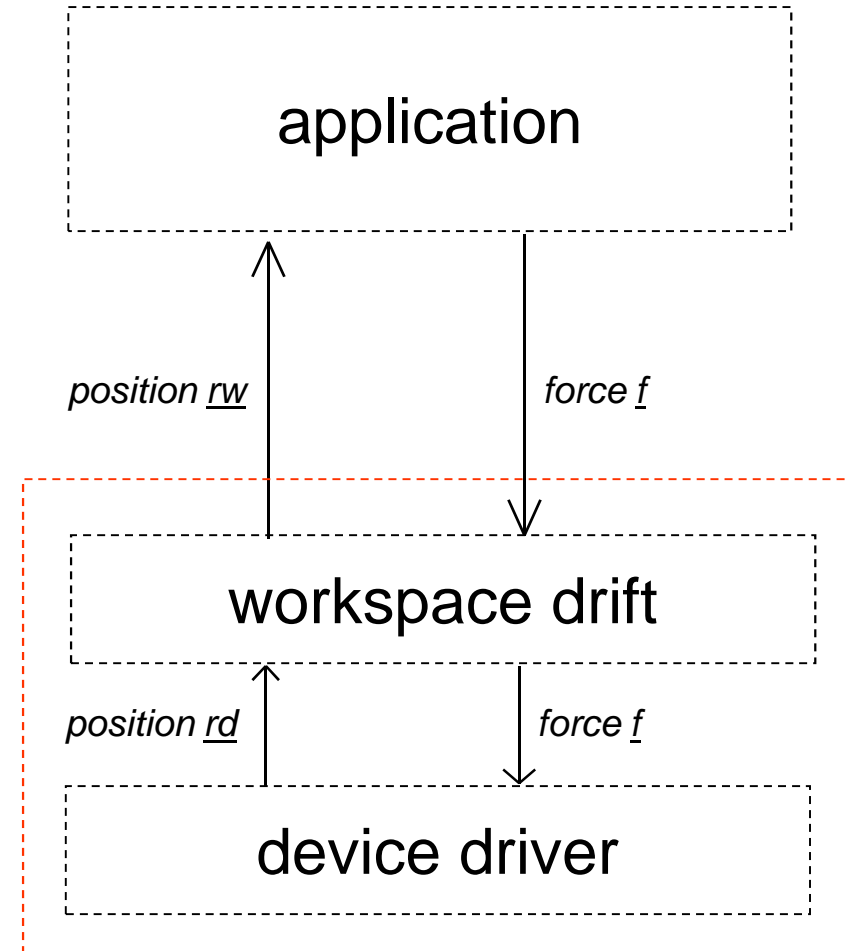
$$\vec{v}_{wd} = k \cdot |\vec{v}_d| \cdot (\vec{r}_d - \vec{r}_{d0})$$

- A workspace drift occur when the device reaches the boundaries of its physical workspace.
- A workspace drift should only occur when the user is moving his/her hand.

Workspace Drift Control



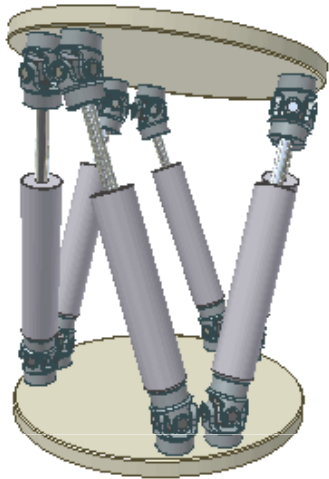
www.chai3d.org



A list of tricks that we will cover

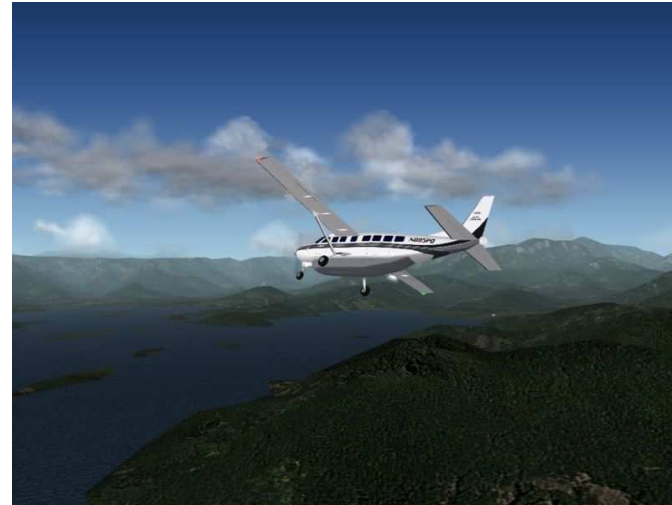
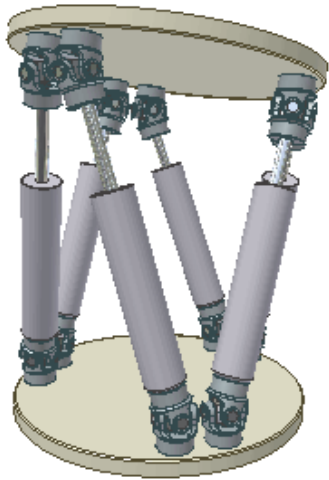
- Rendering 3D shapes using 2 DOFs
 - i.e. how to project positions and forces on smaller rank vectorial spaces
- Rendering 2D shapes using 1 DOF
 - i.e. how work can be your ally (and your enemy)
- Rendering small bumps to feel **large**
 - i.e. how our sensitivity to force direction is not that good
- Rendering **large** virtual environments using small devices
 - i.e. how to take advantage of humans' poor perception of position
- **Rendering fast cars without moving much**
 - **i.e. our vestibular sense is also pretty limited**

Rendering fast cars without moving much



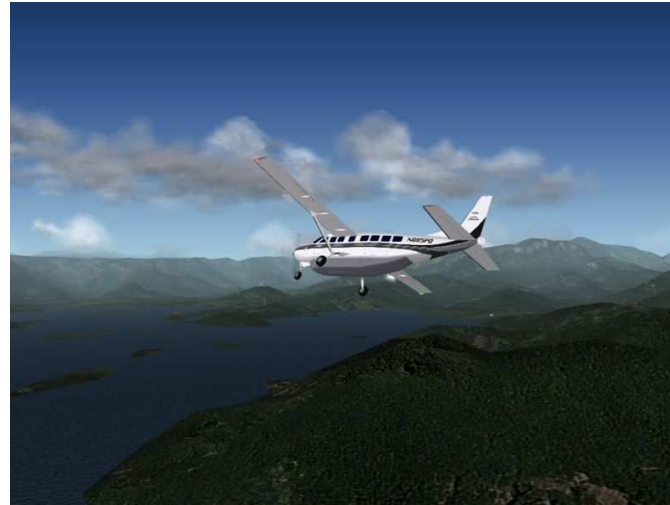
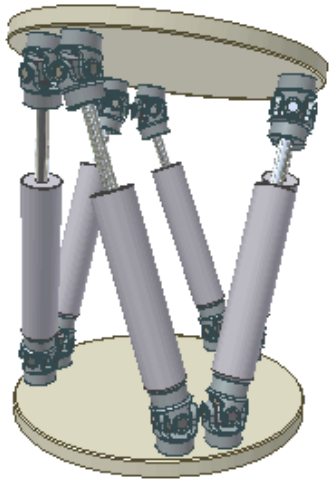
Rendering fast cars without moving much

- Basic issue:
 - Platform (left) has limited workspace, plane (right) doesn't



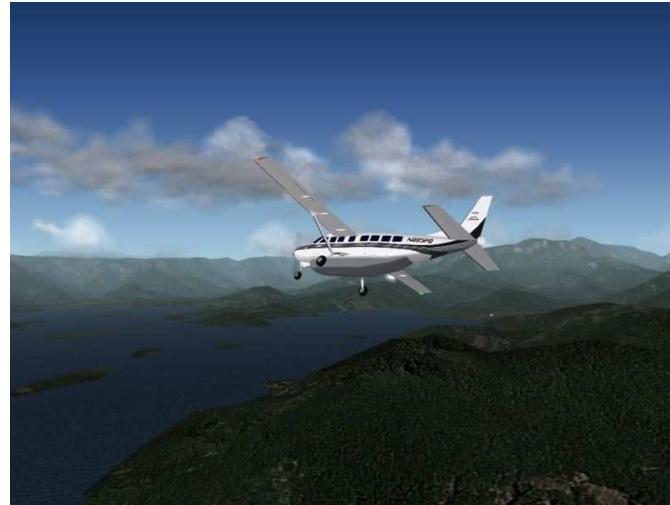
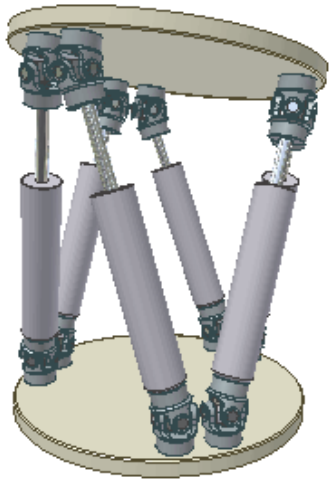
Rendering fast cars without moving much

- Washout filters trick the user's vestibular sense by
 - Splitting accelerations into low and high frequency components
 - Applying high frequency components directly (as they require limited workspace)
 - Applying low frequency components by taking advantage of gravity (tilting the platform)



Rendering fast cars without moving much

- Why does this work?
 - Platform tilting is not perceived if it happens slower than 3 DEG/sec



References

Hayward on tactile tricks

<http://www.cim.mcgill.ca/~haptic/pub/VH-BRB-08.pdf>

Srinivasan on visual and haptic cues in stiffness perception

http://touchlab.mit.edu/publications/1996_006.pdf

Zilles and Salisbury on god-object rendering, friction and force shading

<http://www.stanford.edu/class/cs277/schedule/assets/Zilles1995.pdf>