Math for Game Programmers:
Juicing Your Cameras with Math

Squirrel Eiserloh
Indie Game Programmer / Designer
Lecturer, SMU Guildhall

A note on .GIF animations

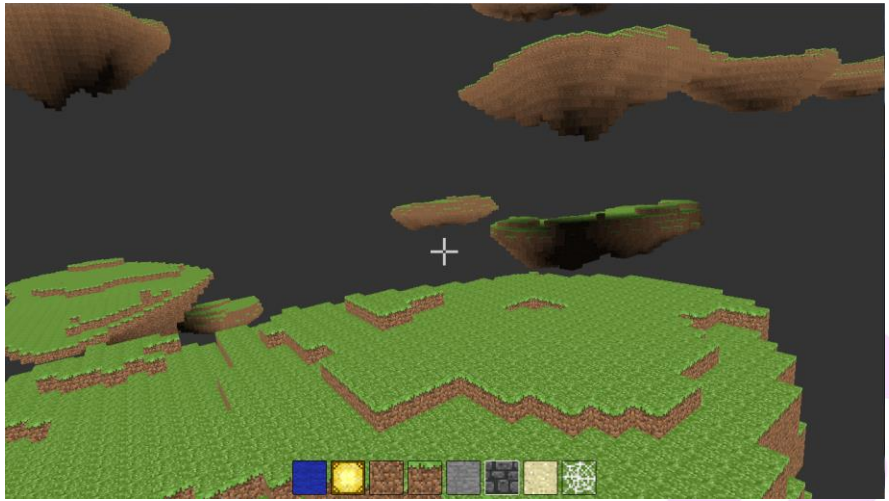
Note: These slides are filled with animated .GIF images, which unfortunately do not animate in this .PDF rendering.

You can download the original PowerPoint slides at:
www.EssentialMath.com/tutorial.htm

Or feel free to contact me via email (squirrel@eiserloh.net)
or Twitter (@SquirrelTweets)!

Overview

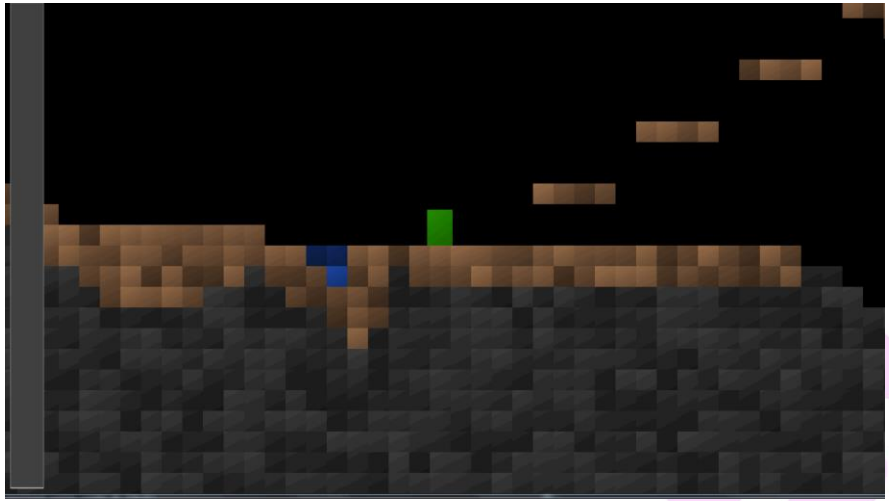
- Camera Shake
  - Translational vs. Rotational
  - Noise vs. Random
  - 2D vs. 3D



Overview

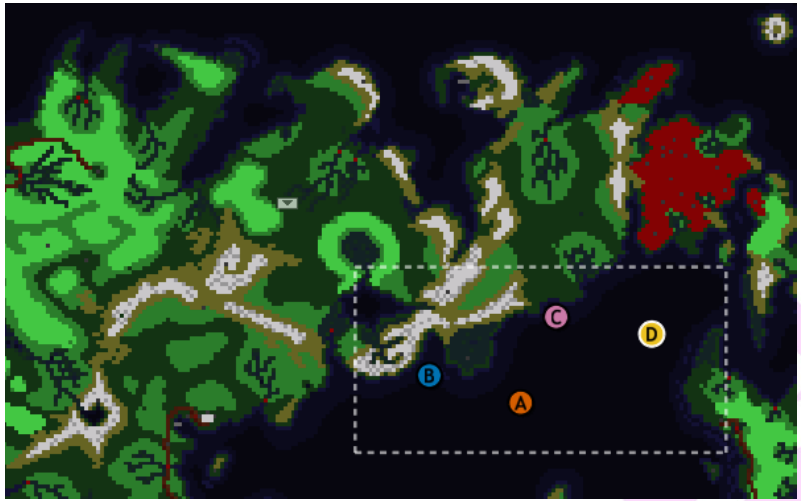
- Camera Shake
- Smoothed motion
  - Parametric motion*
  - Asymptotic Averaging
  - Asymmetric Asymptotic Averaging

whut?



Overview

- Camera Shake
- Smoothed motion
- Framing
  - Points of focus
  - Points of interest
  - Feathering

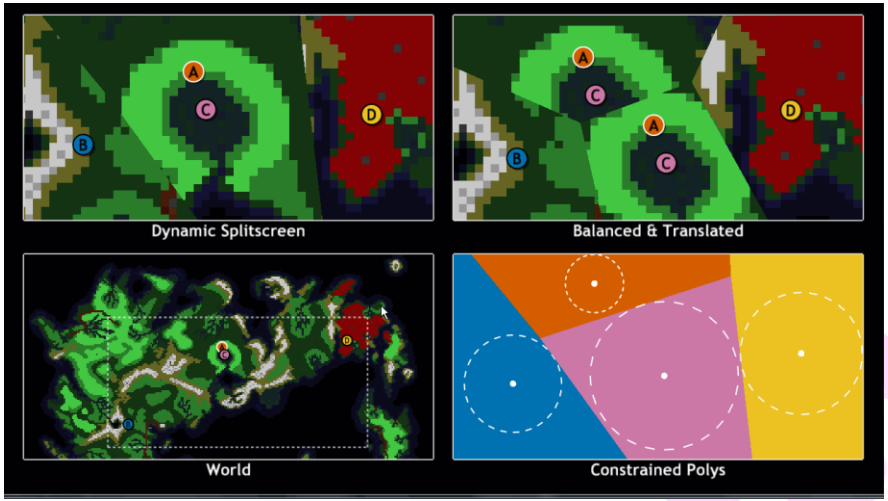


Overview

- Camera Shake
- Smoothed motion
- Framing
- Voronoi split-screen
  - Construction
  - Player- vs. split-relative
  - View merging
  - Feathering

Overview

- Camera Shake
- Smoothed motion
- Framing
- Voronoi split-screen
- Tease and a Challenge



Juice is the new black

Jan Willem Nijman - Vlambeer - "The art of screenshake"

Juice it or lose it - a talk by Martin Jonasson & Petri Purho

Juice is the new black

With great power comes great responsibility

Camera shake is like salt

boring

OMG MAKE IT STOP

(demo: Mantis trauma)

Camera shake

- Maintain a "trauma" level in [0,1]
- Damage, stress adds trauma (+= 0.2 or 0.5)
- Trauma decreases (linearly) over time
- Camera shake is **trauma^2** or **trauma^3**
  - Why does this feel **right**? (spring and damper!)
  - Why does this feel **good**? (escalations perceptible!)

Trauma .30, .60, .90 means 3%, 22%, 73% shake

(demo: Mantis rot, trans, both)

Camera shake: translational vs. rotational

In 2D:

- **Rotational** feels okay, but kinda lame
- **Translational** feels nice
- **Translational + Rotational** = Awesome

But what about in 3D?

(demo: SimpleMiner trans, rot, both, BAD)

Camera shake: translational vs. rotational

In 2D:

- **Rotational** feels okay, but kinda lame
- **Translational** feels nice
- **Translational + Rotational** = Awesome

In 3D:

- **Translational**: super lame! (why?)
- **Rotational**: nice!
- **Translational**: VERY BAD (why?)

Camera shake: in VR


+

=


With great power comes great responsibility

Camera shake: implementation

In 2D,

compute shake angle and offset:

- angle = maxAngle * shake * GetRandomFloatNegOneToOne();
- offsetX = maxOffset * shake * GetRandomFloatNegOneToOne();
- offsetY = maxOffset * shake * GetRandomFloatNegOneToOne();

then add it to the camera for that frame (preserve the base camera)

- shakyCamera.angle = camera.angle + angle;
- shakyCamera.center = camera.center + Vec2(offsetX, offsetY);

Camera shake: implementation

In 3D, same thing:

- yaw = maxYaw * shake * GetRandomFloatNegOneToOne();
- pitch = maxPitch * shake * GetRandomFloatNegOneToOne();
- roll = maxRoll * shake * GetRandomFloatNegOneToOne();
- offsetX = maxOffset * shake * GetRandomFloatNegOneToOne();
- offsetY = maxOffset * shake * GetRandomFloatNegOneToOne();
- offsetZ = maxOffset * shake * GetRandomFloatNegOneToOne();

(actually, there's a better way... wait for it...)

Camera shake: random vs. smoothed noise

Use Perlin noise instead!

- yaw = maxYaw * shake * **GetPerlinNoise**(seed, time, ...);
- pitch = maxPitch * shake * **GetPerlinNoise**(seed+1, time, ...);
- roll = maxRoll * shake * **GetPerlinNoise**(seed+2, time, ...);
- offsetX = maxOfs * shake * **GetPerlinNoise**(seed+3, time, ...);
- offsetY = maxOfs * shake * **GetPerlinNoise**(seed+4, time, ...);
- offsetZ = maxOfs * shake * **GetPerlinNoise**(seed+5, time, ...);

(demo: Mantis, Perlin vs. Random)

Camera shake: random vs. smoothed noise

Smoothed fractal (e.g. Perlin) noise is **WAY** better than random for screen shake. Why?

- Smoothed noise feels better
- Smoothed noise automagically works with pause and slow-motion
- Smoothed noise has adjustable frequency
- Smoothed noise is more easily reproducible on replay
- etc.

Takeaways

- Camera shake = trauma^2 (or t^3)
- 2D: translational + rotational
- 3D: rotational only
- Tread carefully in VR!
- Use Perlin noise for shakes

  and for, like, everything else. Seriously!

Smoothed motion

- We often want the camera to follow the player.
- Player movement is often erratic, or jerky!
- Smoothed motion to the rescue.
- Best approach: consider use of cubic Hermite curves (see: "Interpolations and Splines" from the GDC 2012 Math Tutorial)
- Or use a simple tool: **Asymptotic Averaging**

Smoothed motion

Juice it or lose it - a talk by Martin Jonasson & Petri Purho

```
x += (target – x) * .1;
```

or

```
x = (.90*x) + (.10*target);
```

Asymptotic Average

```
x = (.90*x) + (.10*target);
```

says

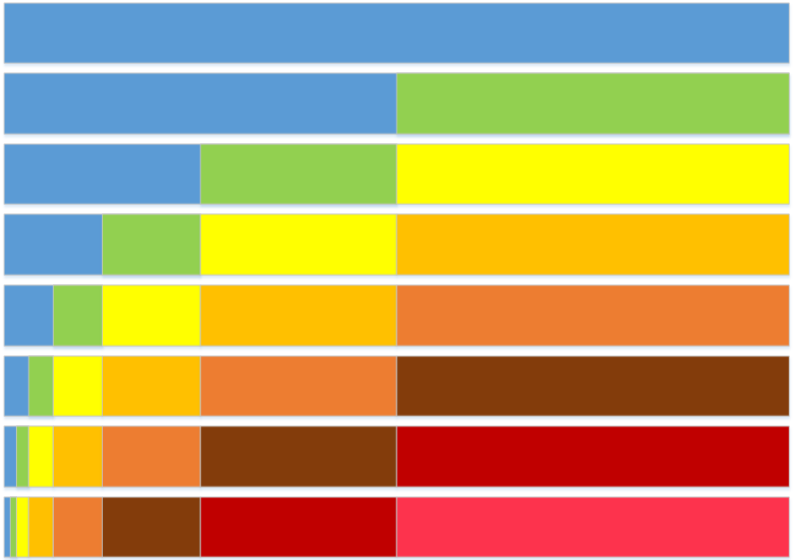
"Each frame we take a 90/10 blend of ourselves and our target."

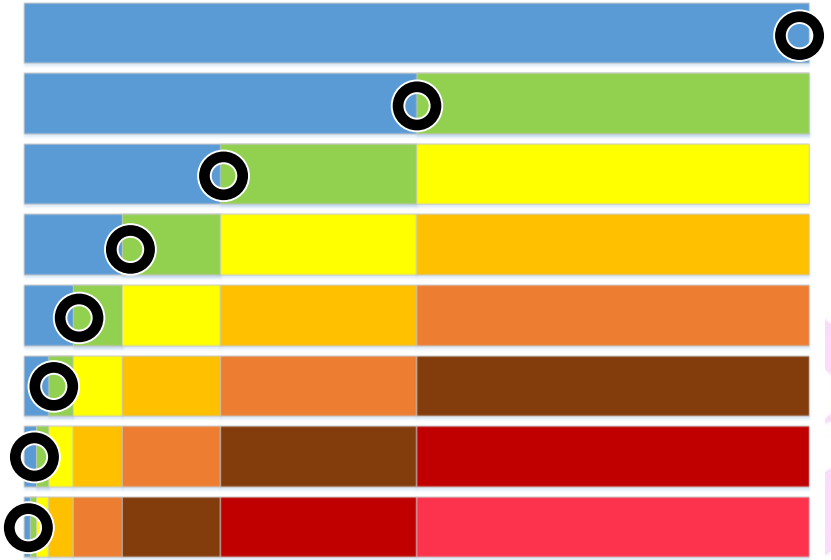
```
x += (target – x) * .1;
```

says

"Each frame, we move 10% closer to our target."

Asymptotic Average





(demo: Mantis, Asymptotic slow, medium, fast)

Asymptotic Average

How fast it moves depends entirely on the weight.

We're talking in the ballpark of:

0.01 = nice and slow (at 60 FPS)

0.1 = reasonably fast

0.5 = incredibly fast

"**Asymptotic**" because it never actually arrives!

(demo: Mantis, Asymptotic custom 4,5)

Asymmetric Asymptotic Average

Also, nothing says **horizontal** and **vertical** camera motion need to be designed the same.

Nor does **upward** movement need to be governed by the same rules as **downward** movement.

The Asymptotic weights can even be non-constant!

Asymptotic Average: one last fix

- This doesn't pause or timescale well!

```
x += (target – x) * .1 * timeScale;
```

...but we can hack around it by scaling weight times **timeScale**:

e.g. 0=paused, .1=slow motion

Takeaways

- Camera shake = trauma^2 (or t^3)
- 2D: translational + rotational
- 3D: rotational only
- Tread carefully in VR!
- Use Perlin noise for shakes

  and for, like, everything else. Seriously!

- Asymmetric Asymptotic Averaging

Framing

Q: What is at the epicenter of our attention?

A_1: Generally, the player.

A_2: Or, at least, the player had better not ever leave the screen (while we're controlling her).

Framing: points of focus

Points of focus are key items which demand a high amount of attention:

- **Primary points of focus**, like **the player**, should never go out of view.
- **Secondary points of focus**, like a specific **targeted enemy**, should not leave the view if possible.

Framing: points of interest

Points of interest, on the other hand, are items which would prefer to be in view, if possible, all other needs being met.

- Points of interest can cause the camera to frame its focal points with a different bias
- Might shift to allow something just offscreen to be seen
- Might shift to draw your attention to something off-center

Framing: points of interest

You can subtly highlight many things with points of interest:

- Enemies
- Loot
- Buttons and levers
- Secret doors
- Traps
- Markers left by level designer (or procedural generator!)

Framing: soft and fuzzy

Feather influences to avoid sudden changes.

Generally, compute “proximity” to each point of interest:

- those outside a threshold have proximity 0
- those inside an inner threshold have proximity 1
- those within the inner & outer thresholds get ~[0,1]
- weight of each point of interest = proximity * importance

Framing: multiple primary focus points

Here's the real struggle:

How do you handle multiple mandatory/primary points of focus?

e.g. Gauntlet:

Framing: multiple primary focus points

Q: How do you handle multiple mandatory points of focus?

A_1: Screen cannot advance if a player would fall behind

A_2: Players can move offscreen

A_3: Players die if forced offscreen

A_4: Players teleported back to main group body if offscreen

A_5: Players can "drag" the screen (and other players) along

A_6: Zoom out to encompass everyone

A_7: Split-screen ← this is the only option that doesn't impact gameplay!

Framing: multiple primary focus points

However, split-screens suck in that you give up 50% or 75% of your screen real-estate.

This is especially sucky in co-op games where the **players are mostly together 95% of the time**.

What can be done?

Takeaways

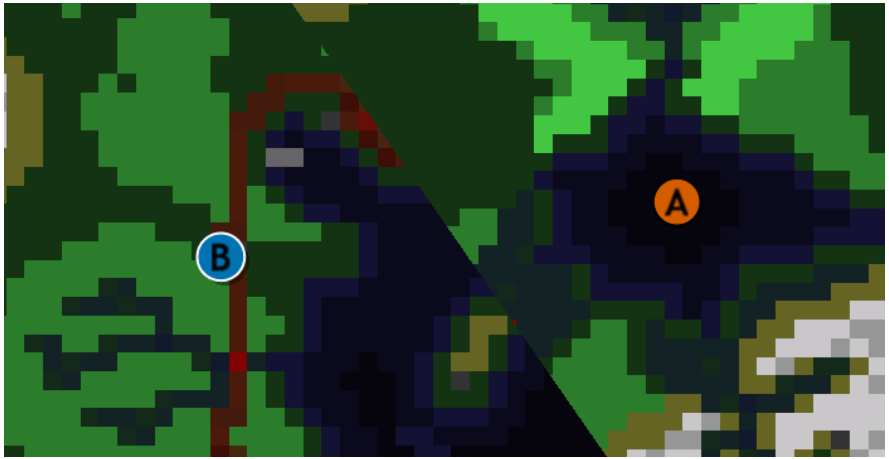
- Camera shake = trauma^2 (or t^3)
- 2D: translational + rotational
- 3D: rotational only
- Tread carefully in VR!
- Use Perlin noise for shakes

  and for, like, everything else. Seriously!

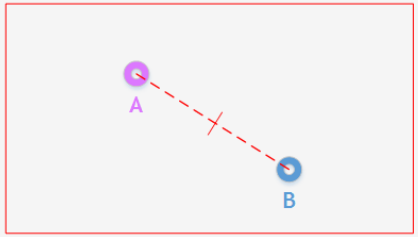
- Asymmetric Asymptotic Averaging
- Blend points of focus & interest
- Use soft feathering everywhere

(demo: Eagle, 2P, shift-1, shift-4)

Voronoi split-screen cameras

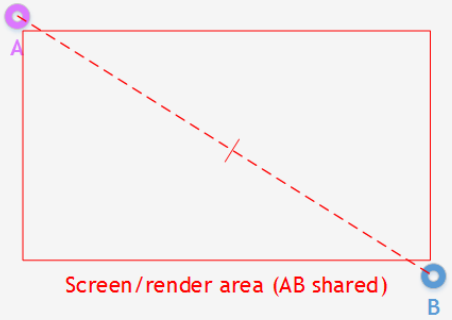


Voronoi split-screen cameras



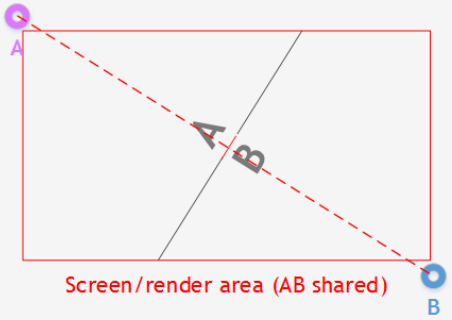
Can players fit onscreen within tolerance?

Voronoi split-screen cameras



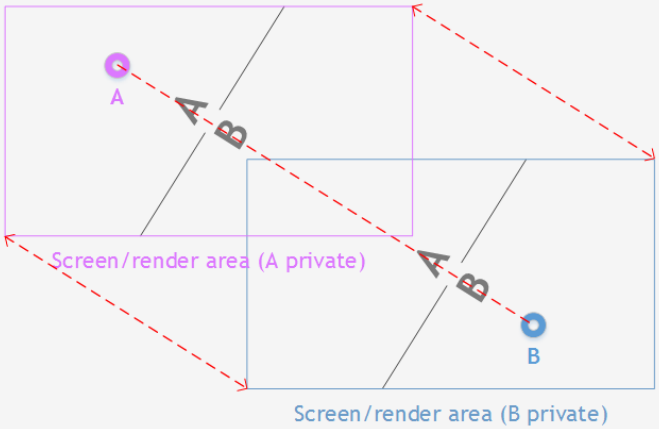
If not…

Voronoi split-screen cameras



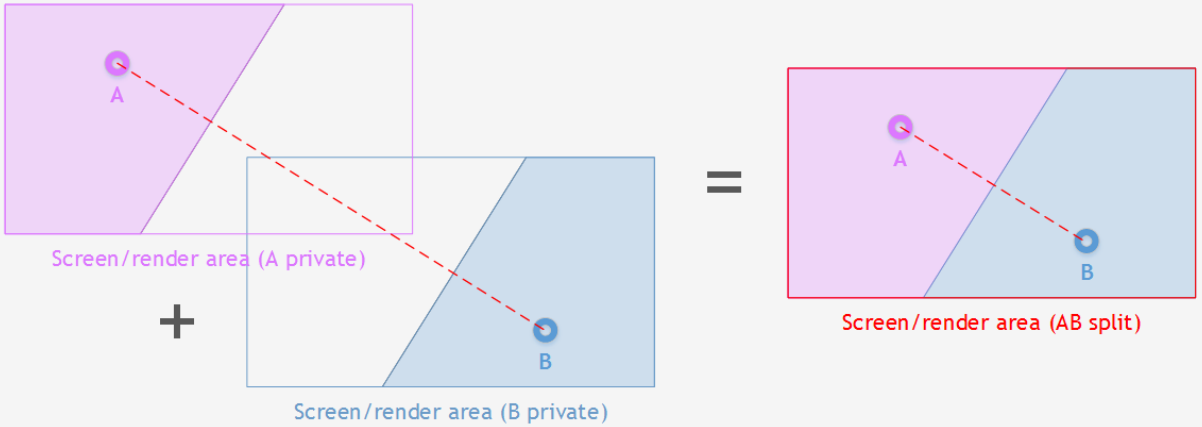
Compute screen-space Voronoi boundary

Voronoi split-screen cameras



Balance private screen spaces on distance

Voronoi split-screen cameras



Algorithm #1: Split and relative positions are true

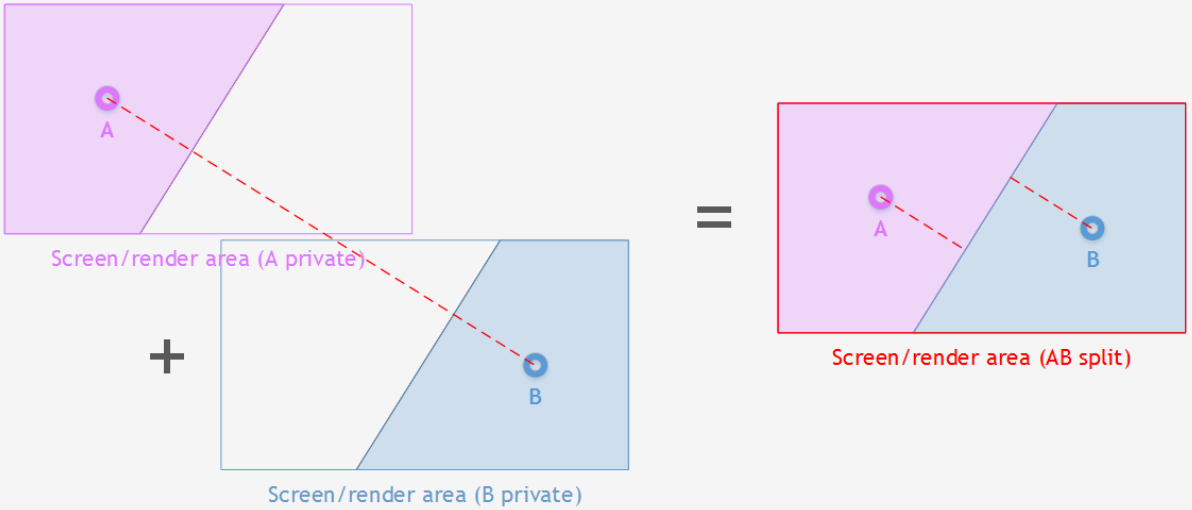
Players are mirrored across split

Pros: To rejoin, players walk toward each other - and toward split

Cons: Players are not centered within their subscreens

Render separately, then stitch

Voronoi split-screen cameras



Algorithm #2: Split is true, but relative positions are not (centered in each subscreen)

Players are mirrored across screen center, but not across split

Pros: Players are centered within their subscreens

Cons: To rejoin, players walk toward split - NOT toward each other

or, recenter regions on players

(demo: Eagle, 2P, Y to disable merge feathering)

Voronoi split-screen cameras

- **Note**: Feathering the transition between merged and separated is crucial!
  - Beyond an "outer" distance, views are fully separate
  - Within an "inner" distance, views are fully merged
  - Between "inner" and "outer", we blend (cross-fade) each view to converge toward the merged view.

(demo: Eagle, 2P, Y to disable merge feathering)

N-way Voronoi split-screen cameras

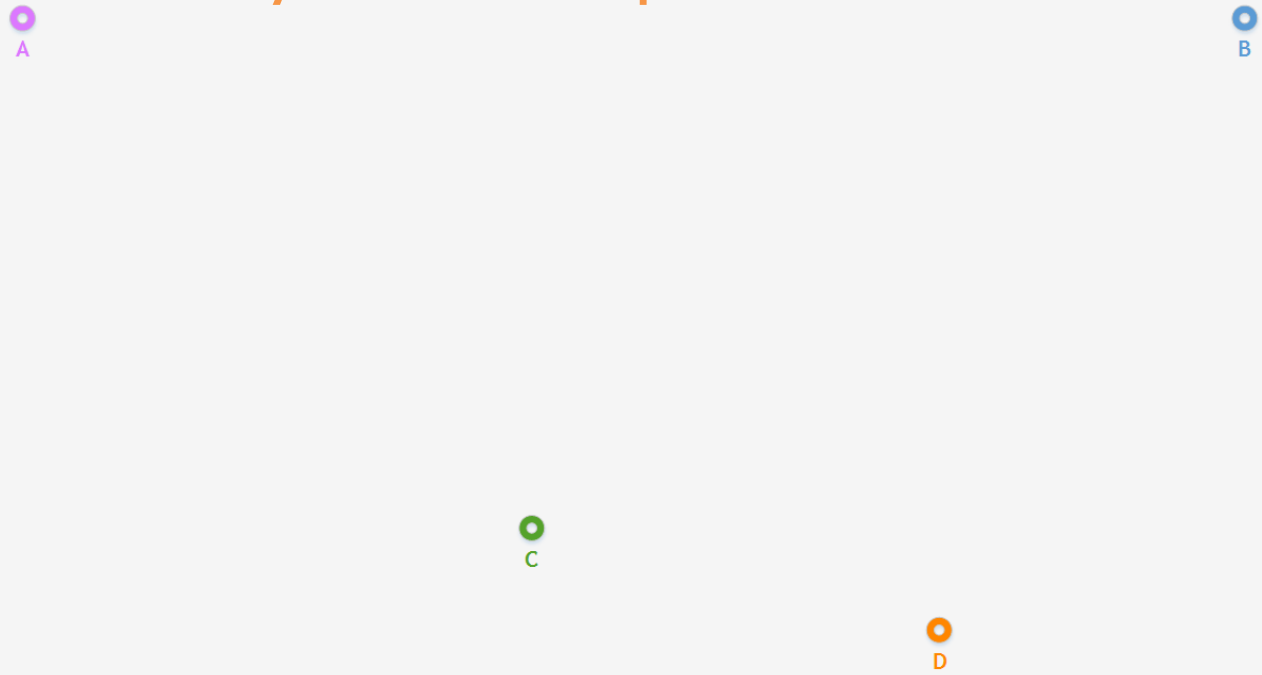
Q: What about 3 or 4 players?

Is it even possible?

A: You betcha!*

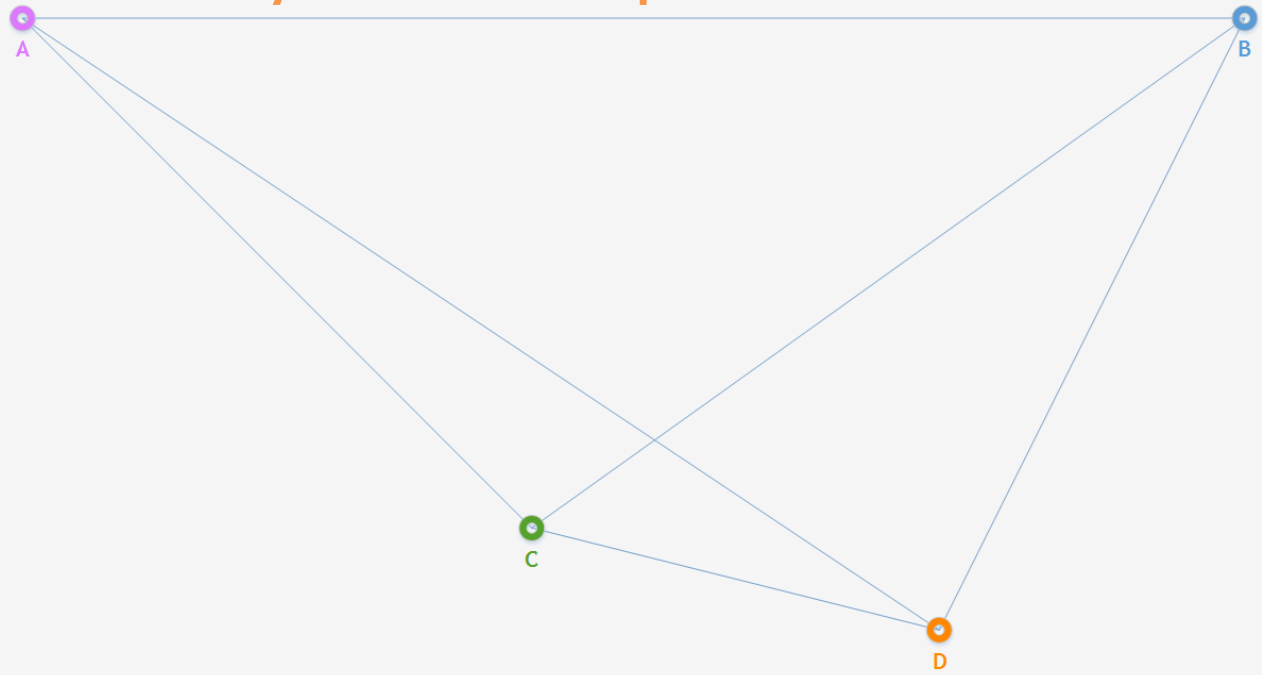
*though there are some tricky bits to navigate

N-way Voronoi split-screen cameras



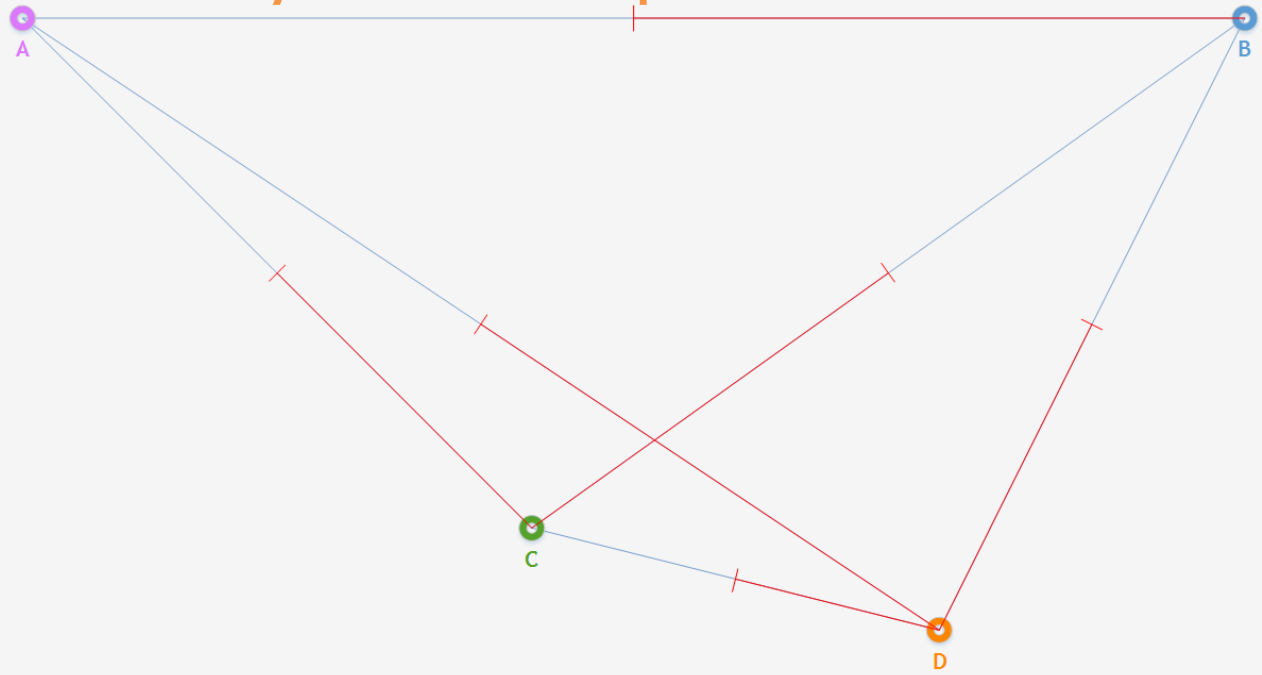
N players in absolute (world) space

N-way Voronoi split-screen cameras



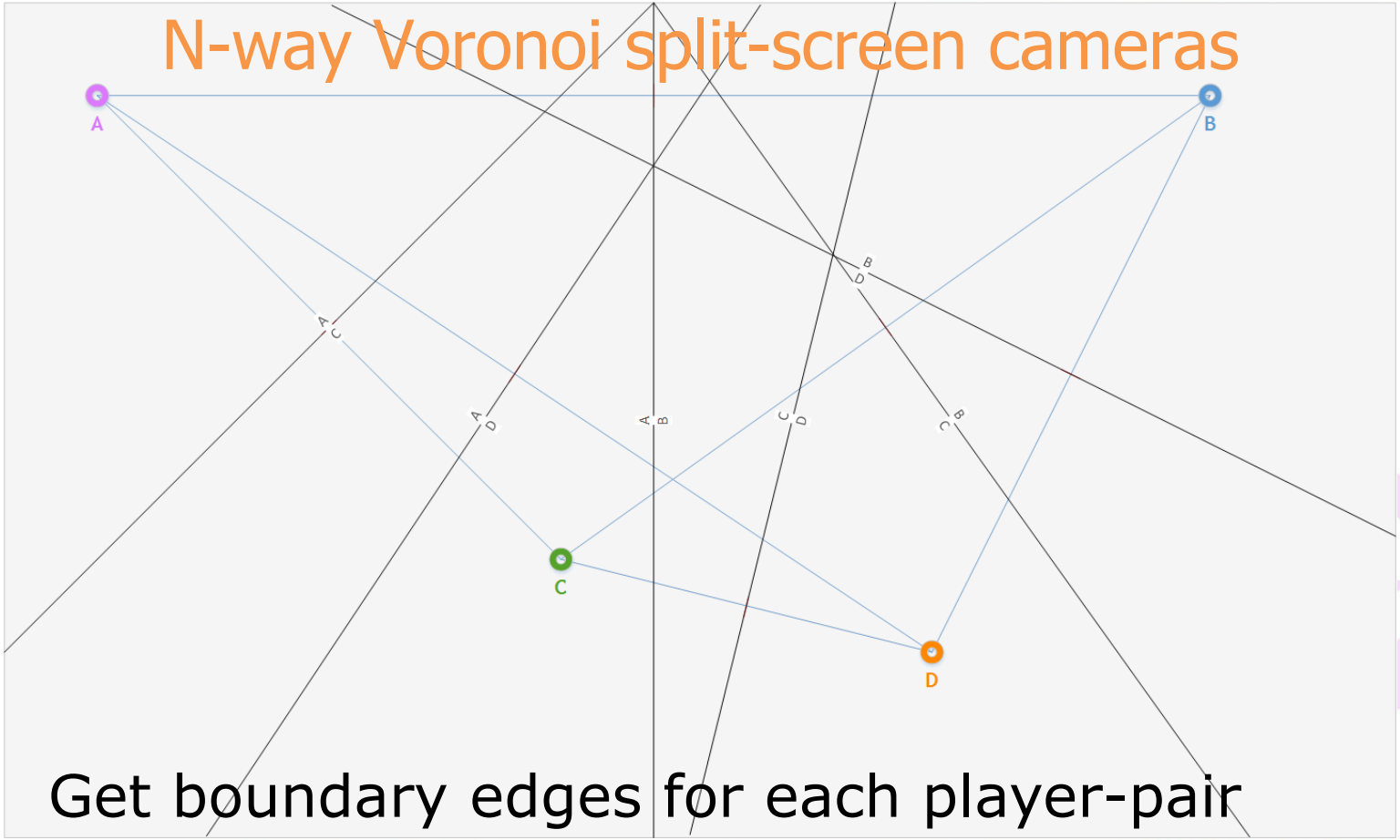
Get relative displacements

N-way Voronoi split-screen cameras



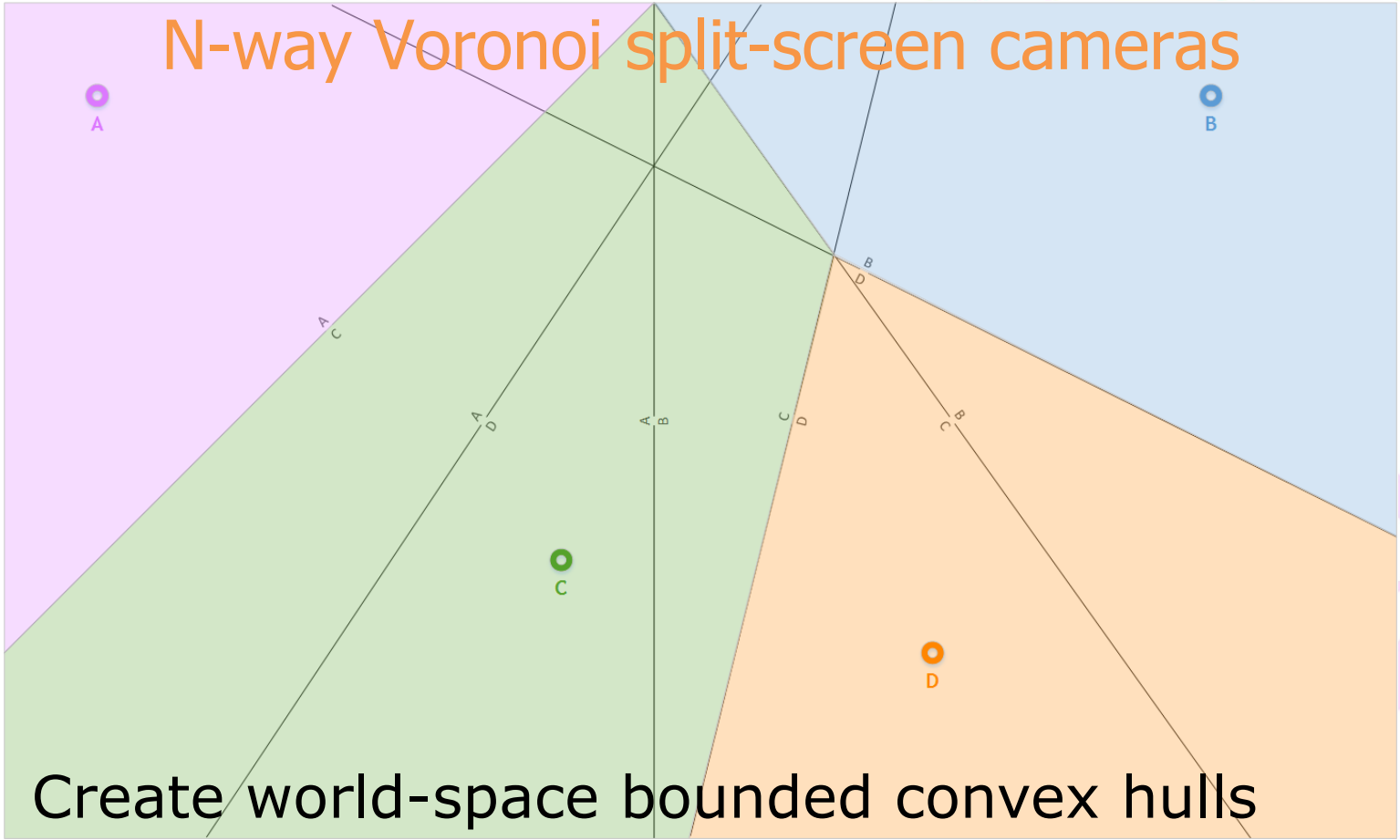
Bisect for midway points & normals

N-way Voronoi split-screen cameras



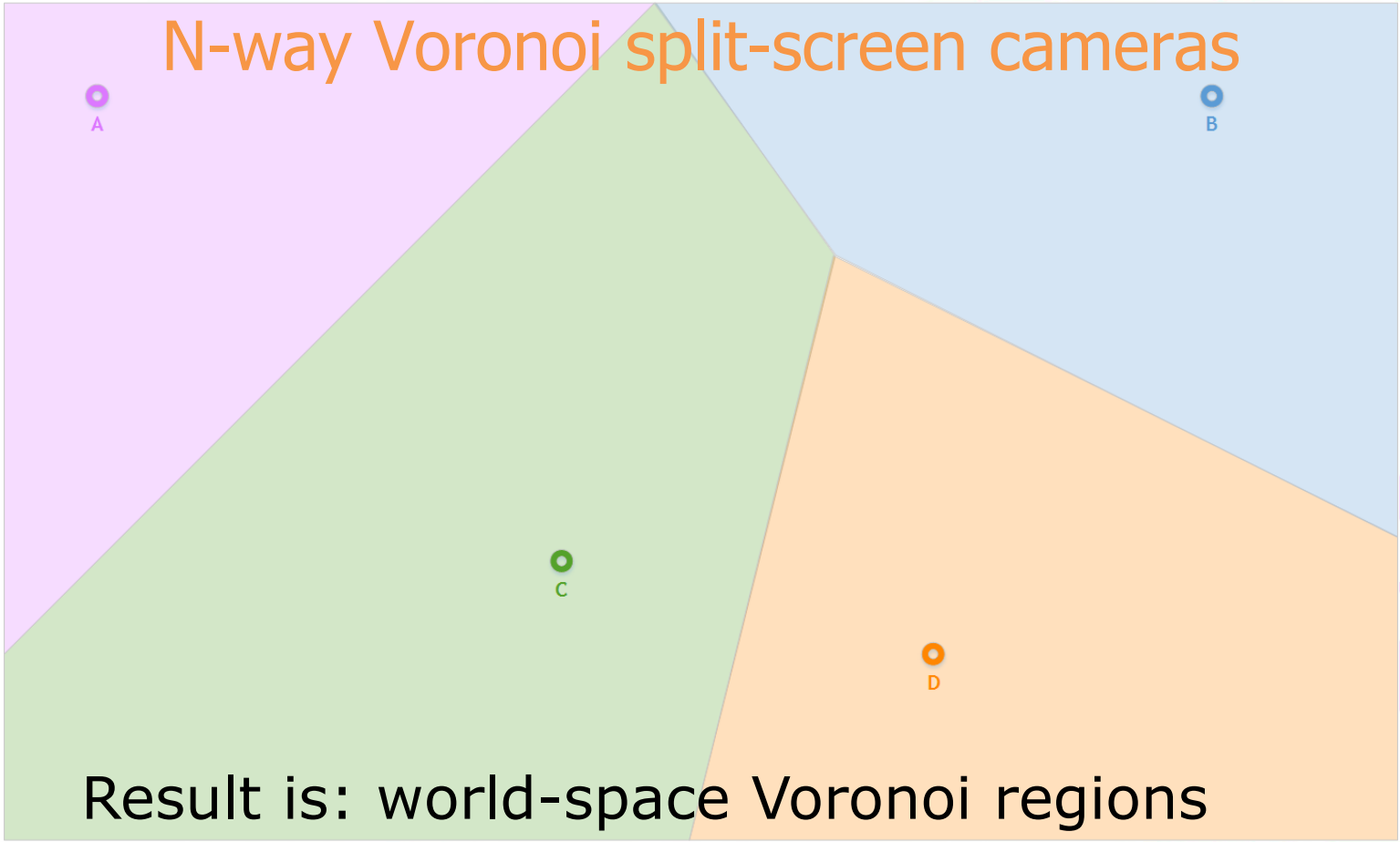
Get boundary edges for each player-pair

N-way Voronoi split-screen cameras

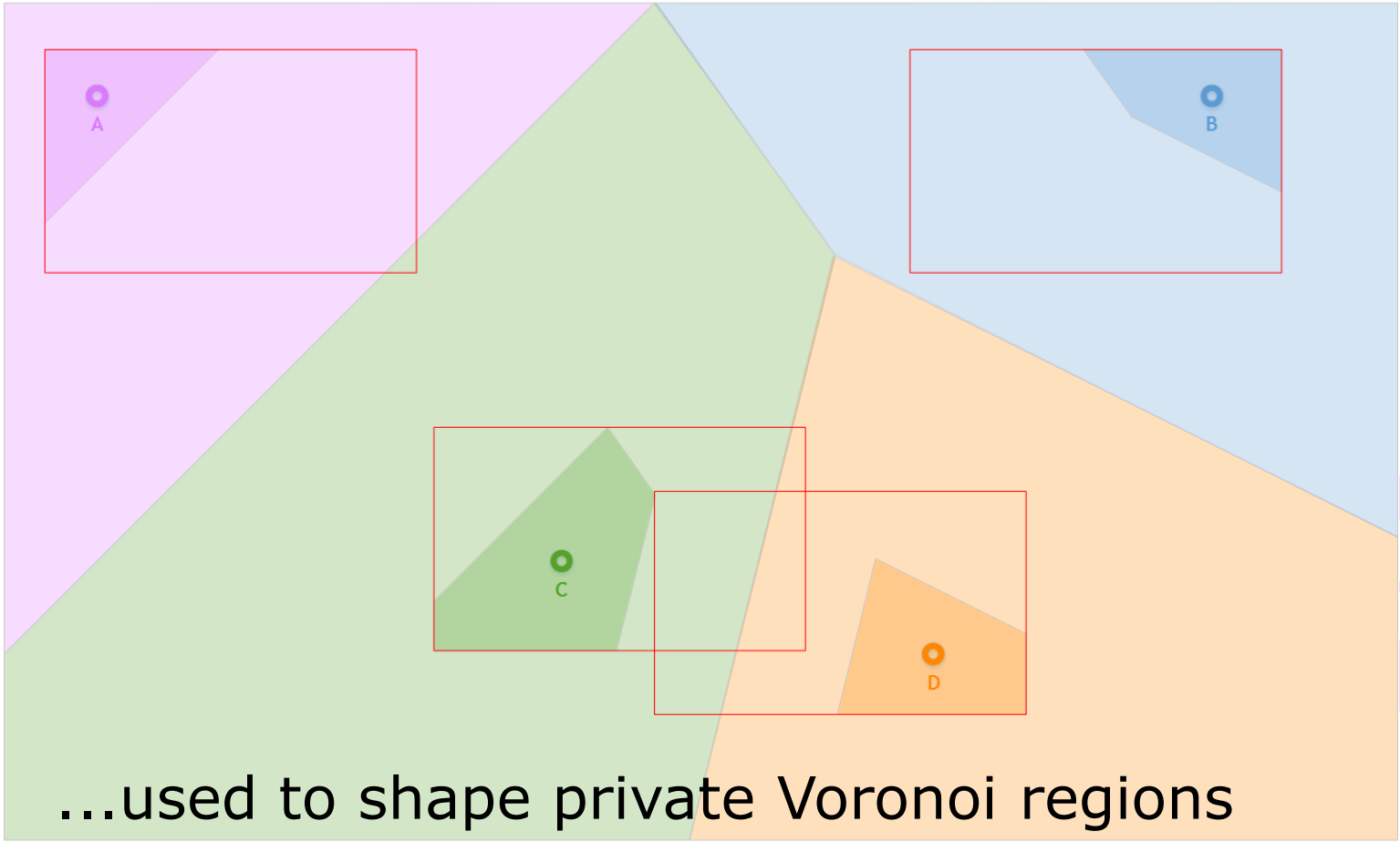


Create world-space bounded convex hulls

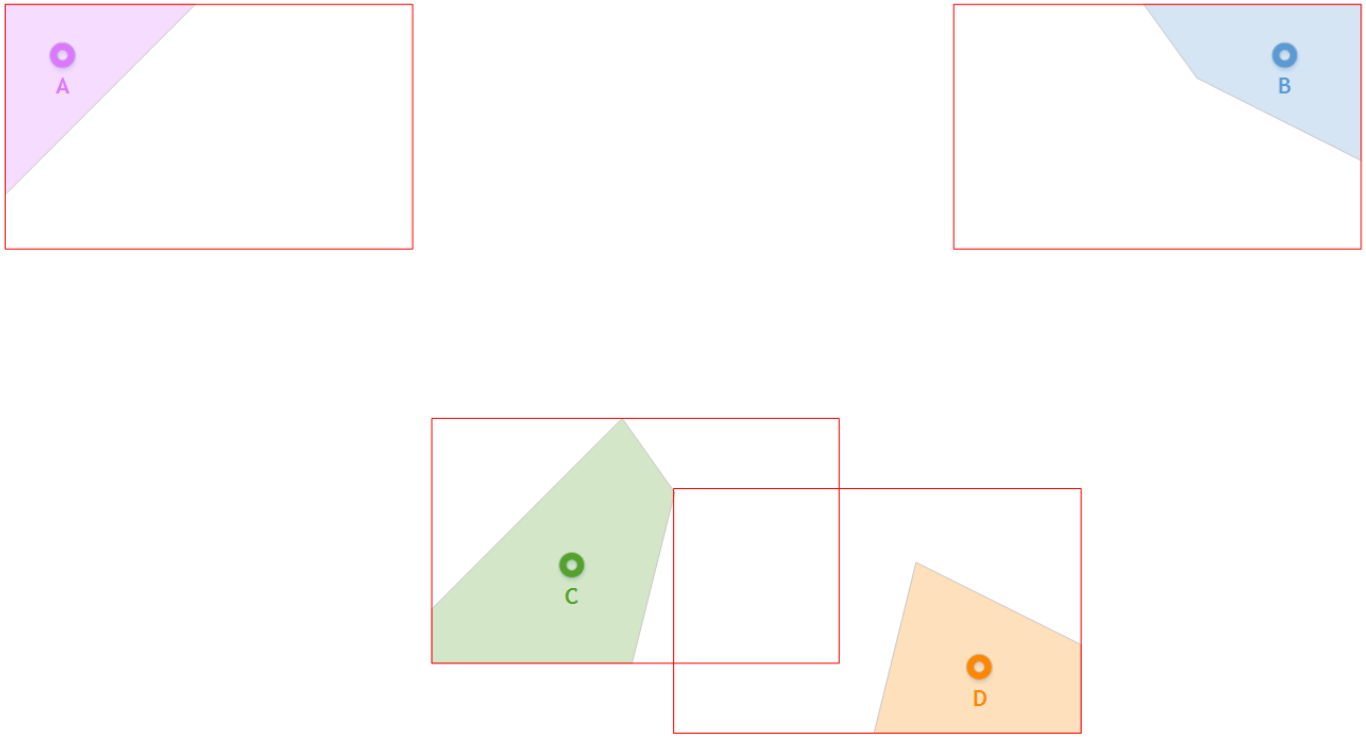
N-way Voronoi split-screen cameras



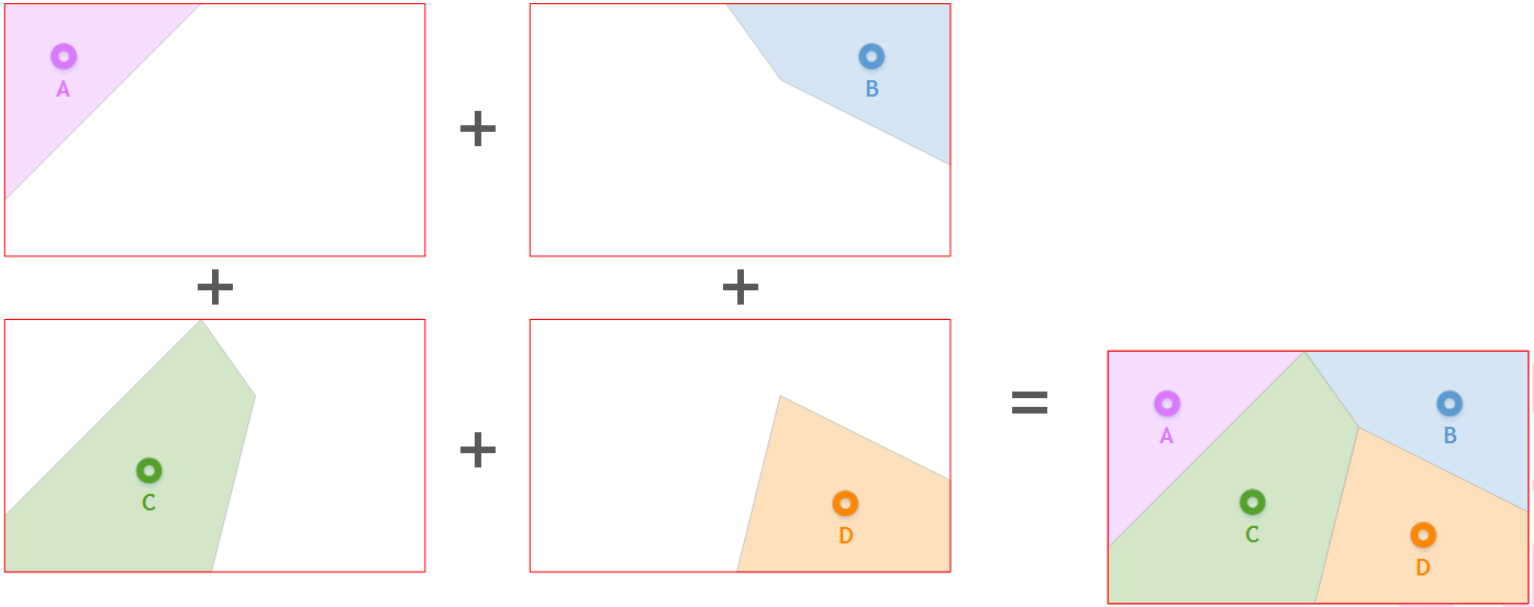
Result is: world-space Voronoi regions



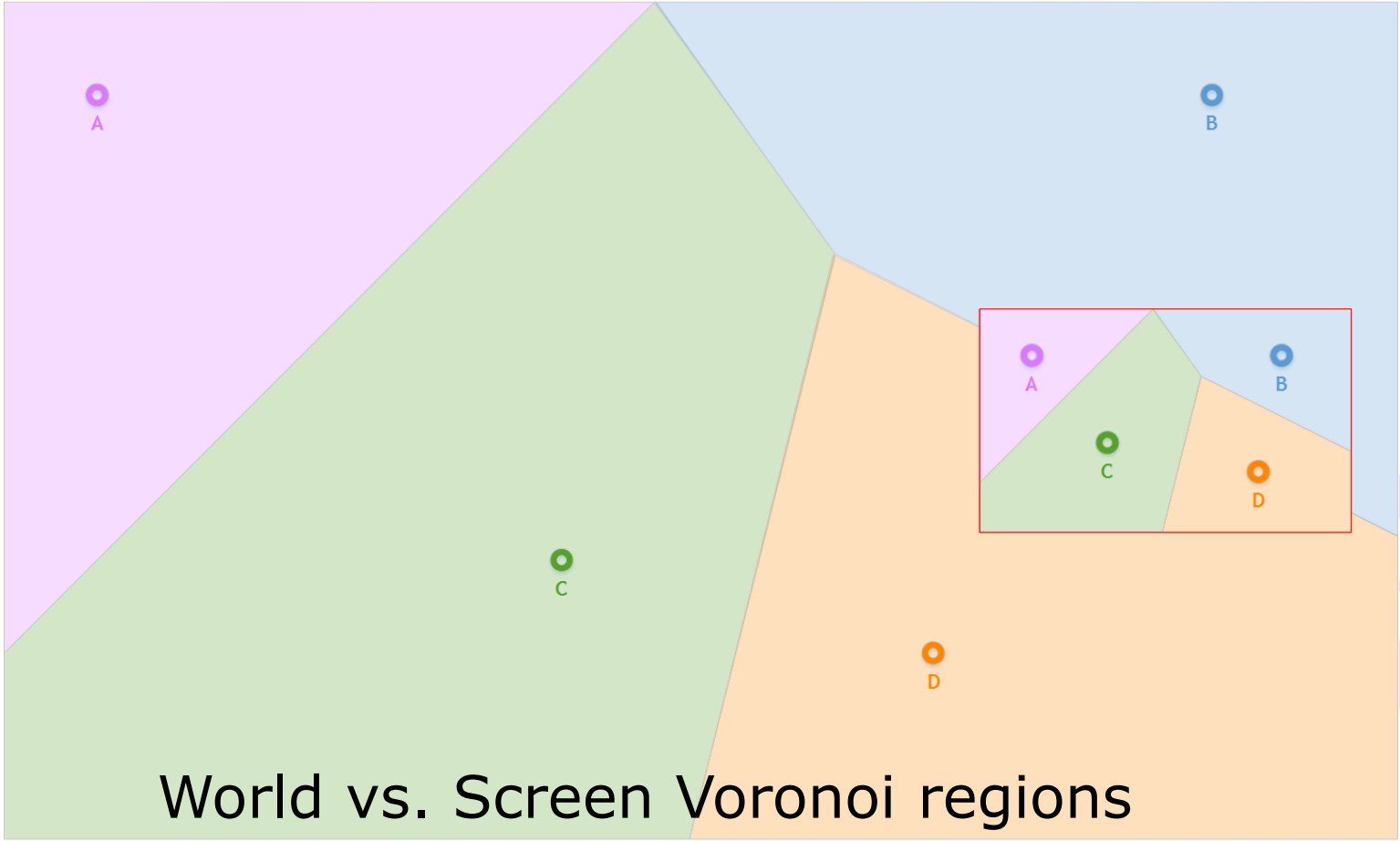
...used to shape private Voronoi regions



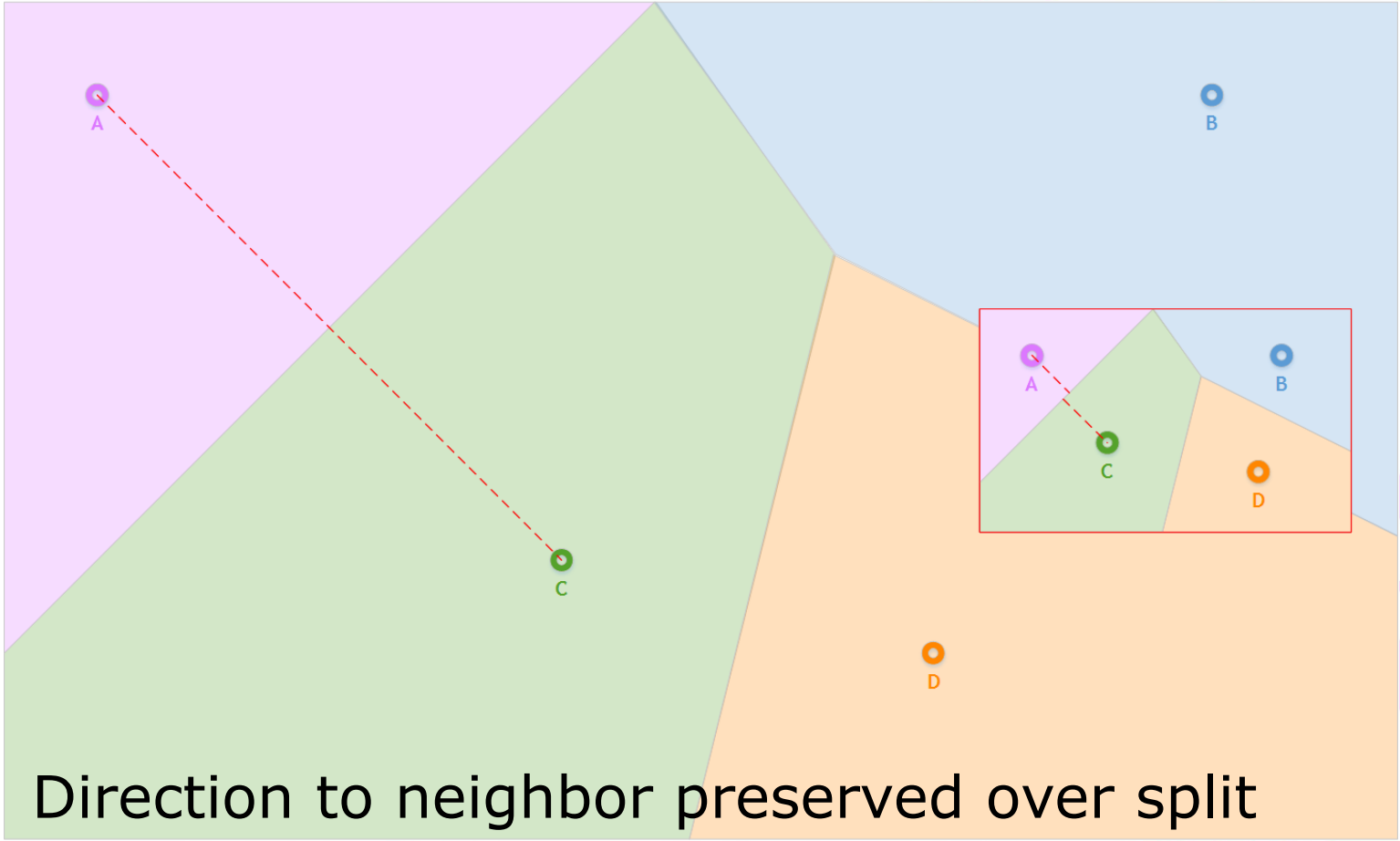
Compute and render separately with stencil



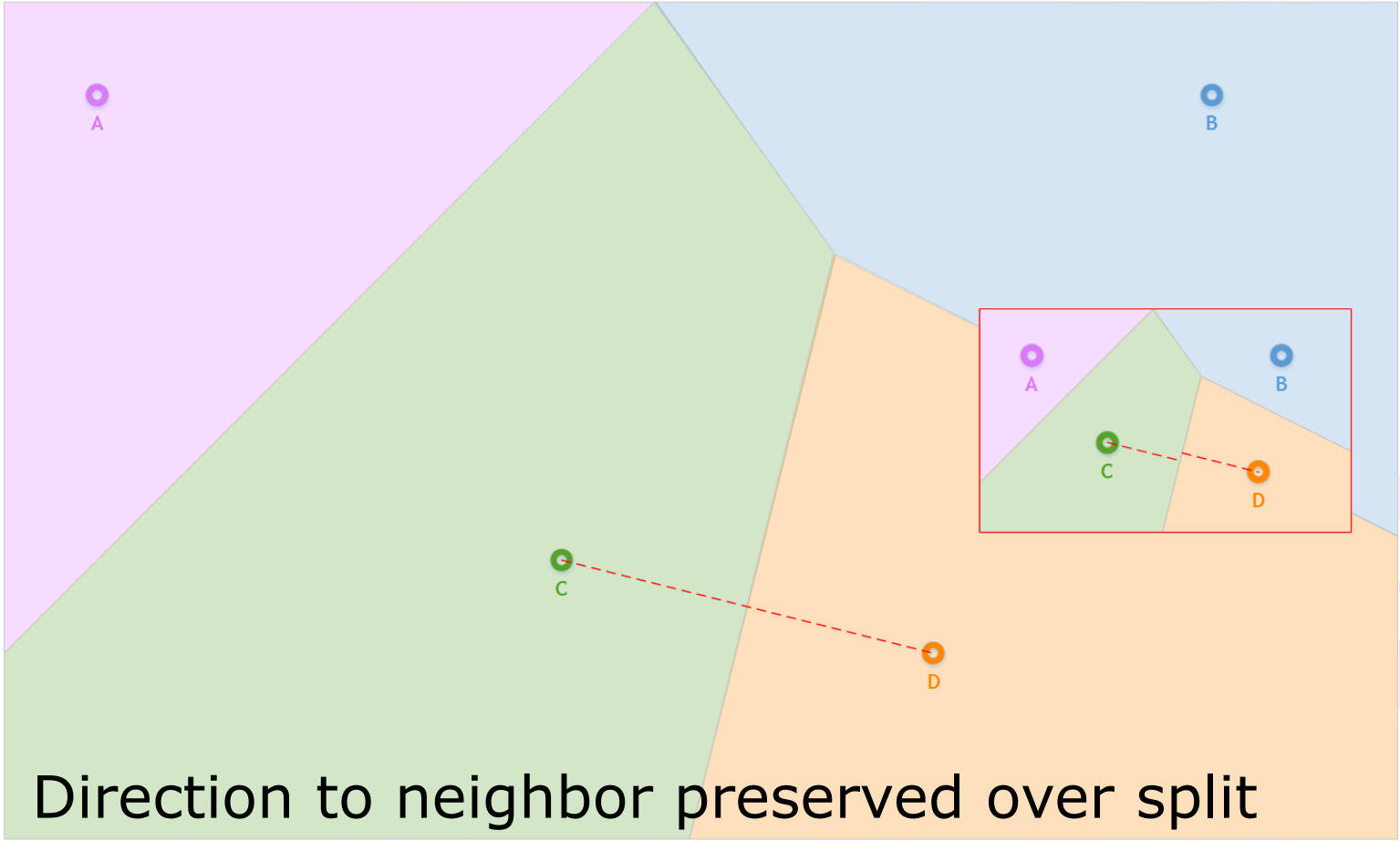
Composite for total view



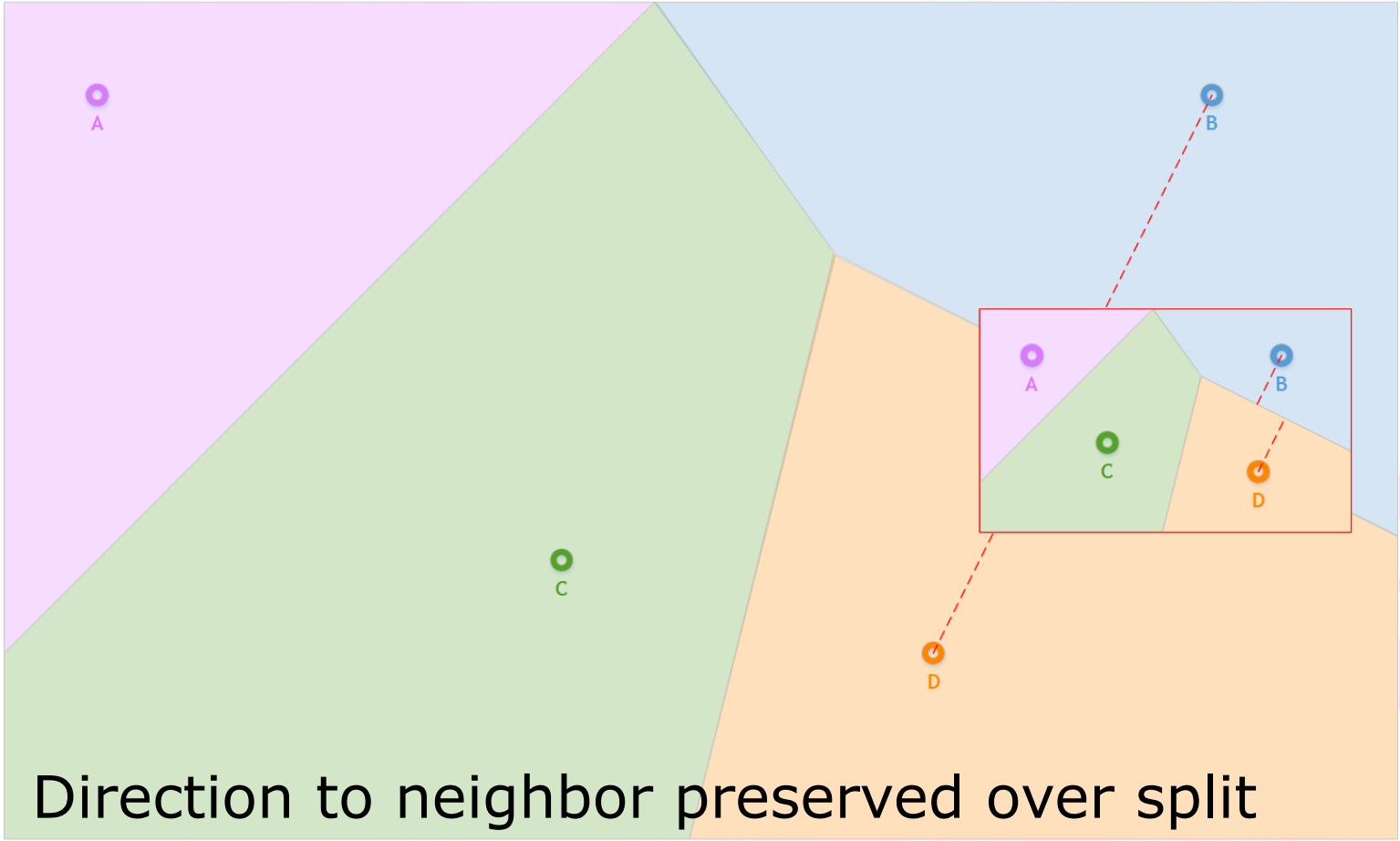
World vs. Screen Voronoi regions



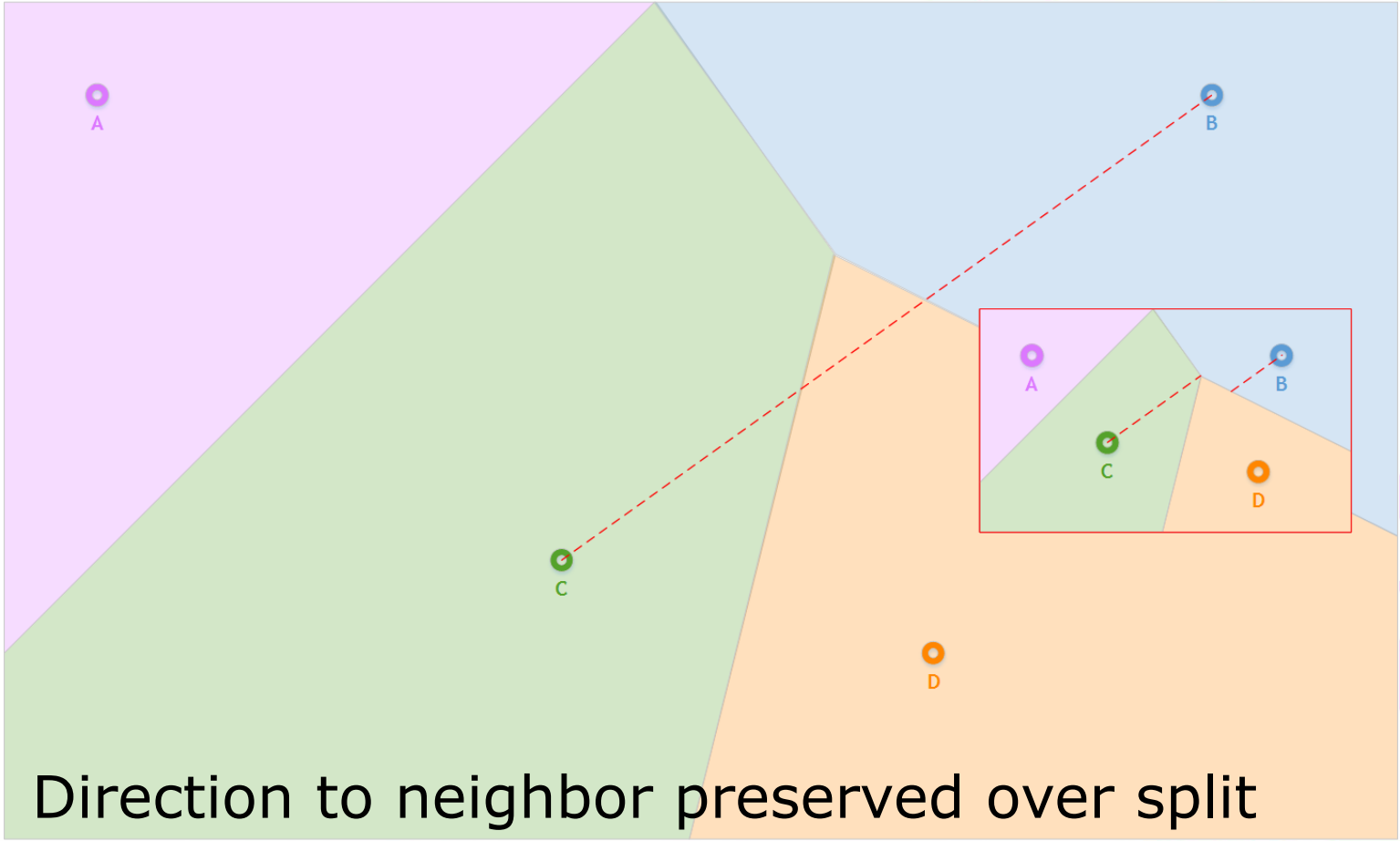
Direction to neighbor preserved over split



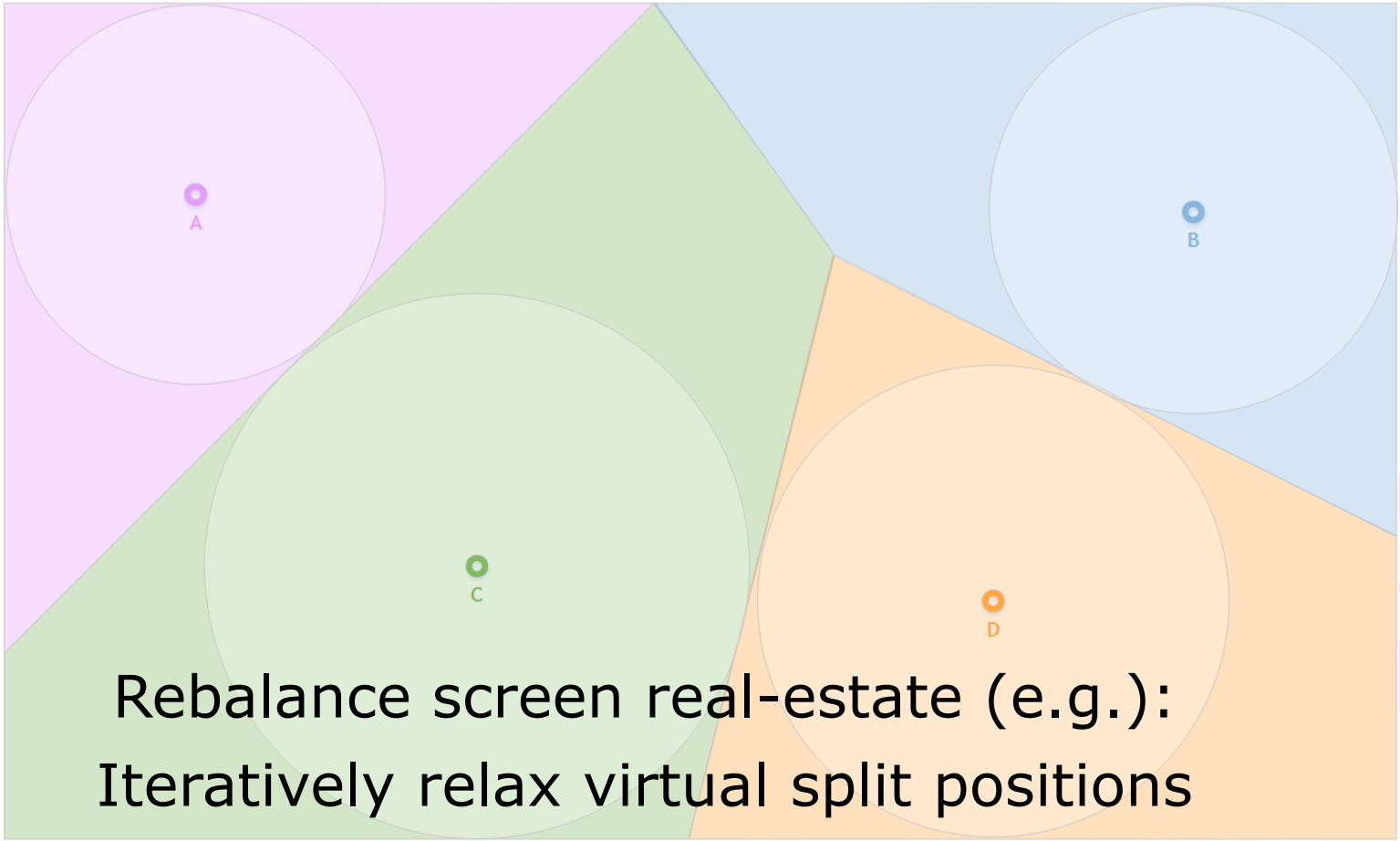
Direction to neighbor preserved over split



Direction to neighbor preserved over split



Direction to neighbor preserved over split



Rebalance screen real-estate (e.g.):

Iteratively relax virtual split positions

Takeaways

- Camera shake = trauma^2 (or t^3)
- 2D: translational + rotational
- 3D: rotational only
- Tread carefully in VR!
- Use Perlin noise for shakes

  and for, like, everything else. Seriously!

- Asymmetric Asymptotic Averaging
- Blend points of focus & interest

- Use soft feathering everywhere
- Consider Voronoi split-screen
- N-way split possible, but tricky
- Use juice liberally, yet wisely
- The camera is a character!
- Check out Itay Keren's article:

  Scroll Back: The Theory and Practice of Cameras in Side-Scrollers
- Check out these Math talks:

  Fast and Funky 1D Nonlinear Transforms (GDC 2015)

  Random Numbers (GDC 2014)

  Interpolations and Splines (GDC 2012)

Takeaways

- Camera shake = trauma^2 (or t^3)
- 2D: translational + rotational
- 3D: rotational only
- Tread carefully in VR!
- Use Perlin noise for shakes

  and for, like, everything else. Seriously!

- Asymmetric Asymptotic Averaging
- Blend points of focus & interest

Thanks!

- Use soft feathering everywhere
- Consider Voronoi split-screen
- N-way split possible, but tricky
- Use juice liberally, but wisely
- The camera is a character!
- Check out Itay Keren's article:

  Scroll Back: The Theory and Practice of Cameras in Side-Scrollers
- Check out these Math talks:

  Fast and Funky 1D Nonlinear Transforms (GDC 2015)

  Random Numbers (GDC 2014)

  Interpolations and Splines (GDC 2012)