• Deliver a sense of speed and action
• Sampling motion for duration of shutter exposure
• Often prohibitively expensive for real-time rendering

Video, start sequence. Zero velocity with full detail texture sampling. Accelerating introduces blur. In the video, see if you can spot glitches on semi-transparent surfaces and disocclusions
Non Real-Time

- Distributed Ray Tracing
  - [Cook et al ’84]
- Frequency Analysis
  - Sheared Reconstruction
  - [Egan et al ’09]

Beautiful, distributed ray traced images.
Frequency analysis with sheared reconstruction filter for efficient sampling kernels, but not real time
Stochastic rasterisation looking promising for upcoming graphics hardware research, but some noise at lower sample rates to address and not accelerated on consoles. McGuire developed a tight bound on stochastic rasterization primitives at HPG 2010. The promise of a high performance motion blur, depth of field, anti-aliasing hardware unit is attractive particularly if programmable.
Accumulation buffer requires high frame rate to avoid ghosting artifacts (see 30fps accumulation buffer comparison video).

Object based methods can also be low cost, but here we need a full scene method for environment and cars in racing game.

Vlachos also presents a frame blur method using just camera motion used in portal, which might be suitable for some racing games where most movement is in the environment rushing past the camera. For Split Second though we wanted multiple object motions to be blurred correctly.
Image space can be costly, and texture space doesn’t handle geometry edges. So, use combination of these methods.
Compute velocities using current pixel depth reprojected with inverse projection matrix from previous frame.

Take a single frame and sample color with a blur kernel along the direction of velocity.

Half speed playback video. Image blur includes texture and geometry.
Motion IDs index array of inverse projection matrices to recover computed velocities.

Motion IDs video.

Multiple depth samples along the blur kernel may be used to validate color samples, clearly at the cost of additional read bandwidth and shader operations. Here we use lower bandwidth motion ID comparisons to separate multiple object motions during the blur kernel calculation.

Motion ids are packed in alpha component of color frame buffer, so the number of taps mentioned after this refers to single texture fetch samples.

We have a higher level algorithm which prioritizes motion ID use, preferring environment and cars, with debris with secondary priority. If motions exceed available ids, they'll revert to the environment id to retain a plausible camera relative blur appearance.
Slow to do one large kernel over screen, so adapt kernel size according to velocities

Console branching hardware is too slow to do this per-pixel, so we do this per tile

For a racing camera, we can statically determine tiles
With a tiled classification of velocities, we can apply this adaptively.

Adaptive tiling video.

Artefacts due to different sampling rates between adjacent tiles are reduced by limiting the difference in adjacent kernel sizes and also conservatively providing for smooth filter coverage for the velocities across tiles.

A recursive filter with multiple passes accumulating results into a successively larger filter (similar to that applied to optimize God Rays method) performs reasonably well with larger velocities, but additional cost for texture resolves and synchronisation between passes just pushed this out of useable range for Split Second.
We can combine with a texture space motion blur, using anisotropic sampling along motion vector for albedo and normal map textures.

This acts as a kind of pre-computed blur kernel supporting the image space blur.

With texture space only, accessing a texture’s mip chain with a single sample equates to accessing a blur kernel computed offline with many samples.

This gives us less than one motion blur sample per pixel, e.g. a simple point with velocity 8 pixels per frame would require 1 sample per pixel on a kernel 8 pixels wide to recover the motion blur without aliasing. So, here those 8 samples are pre-computed and embedded in the mip chain and sampled with a single texture fetch, giving a sampling of less than one sample per pixel.

In fact we combine image space sampling with regular isotropic mip map sampling using a simple mip LOD bias (actually only possible when combining with image space sampling). This leads to some excess lateral blur, but in practice much faster than anisotropic sampling.

On some surfaces we still use anisotropic sampling for geometric orientation filter quality, but bias isotropically rather than computing the full velocity and geometric anisotropic sample gradients. It works out to a simple tex2Dbias op without gradients. In this case, a few more ops for gradient calculations might be worth it.
So, let's look at a performance comparison of the stages of our solution video.

The performance of the final effect is relatively coherent between frames without spikes occurring. This is particularly regular during normal racing camera modes (first person, behind, front, side views at speed) and even with Split Second power plays with explosions and many objects moving around. The render cost of the image space blur will peak when using classified adaptive sampling with maximum velocities covering the screen, at which point it equates to a fixed sampling cost. In such a case, it classifies all tiles to a maximum number of taps over the whole screen due to a screen full of high velocities. In practice though, velocities are clamped at a high level in code to bound this cost.
Comparison

- Accumulation buffer
  - Incremental 2 frame motion blur (50.0ms)
  - Ghosting due to low temporal sampling rate (30fps)
Limitations

• Moving semi-transparencies, shadows & skinning
  – Motion IDs per pixel, multiple velocities per pixel

• Solution
  – Re-order layers
    • draw after blur
  – Clamp velocities

Need multiple motion IDs or velocities per pixel.
Limitations

- Reprojection clipping for velocity calculation
  - Projection singularity
- Clamp velocities
  - Shader clip too slow
  - So scale according to camera scenarios

Clipped projection edge case.
Limitations

• Disocclusions
  – Need to sample visibility over multiple frames

• Solutions
  – Accumulation buffer
    • With reprojection to accelerated multiple coherent frames
  – Temporal stochastic rasterization
  – Visibility confidence weighting
  – Shared shading with multi-visibility sampling (REYES)

• However, low cost/benefit for current action racing games

Sampling a single frame isn’t strictly enough.
Conclusion and Further Work

- Very fast and good quality motion blur
- Future
  - Reprojection frame coherent accumulation buffer
    - High (60+) fps with frame interpolation required
      - Possibly trading reduced quality and scene complexity elsewhere
  - Stochastic rasterization

In addition, for a REYES style architecture, as long as you can emit velocities and depth you can apply a fast screen space sampling in a post processing layer.
Thanks

- Baylor Knight
- Julian Adams
- Jeremy Moore
- Neil Hutchinson
- Kevin Egan
- Jason Green
- Dan Porter
- Stuart Pharoah
- Paul Ayliffe
- Peter-Pike Sloan