Primitive processing and advanced shading architecture for embedded space

Maxim Kazakov Eisaku Ohbuchi
Digital Media Professionals, Inc

Contributions

- Vertex cache-accelerated fixed and variable size primitive processing
  - One-pass on-chip implementation of various geometry processing algorithms
  - Enables geometry reconstruction from compact description
- Configurable per-fragment shading
  - Dot product+LUT machine
  - Various shading models can be mapped
  - On-chip material description
- Reduced memory bandwidth requirements
- Realized in embedded-space architecture

Motivation

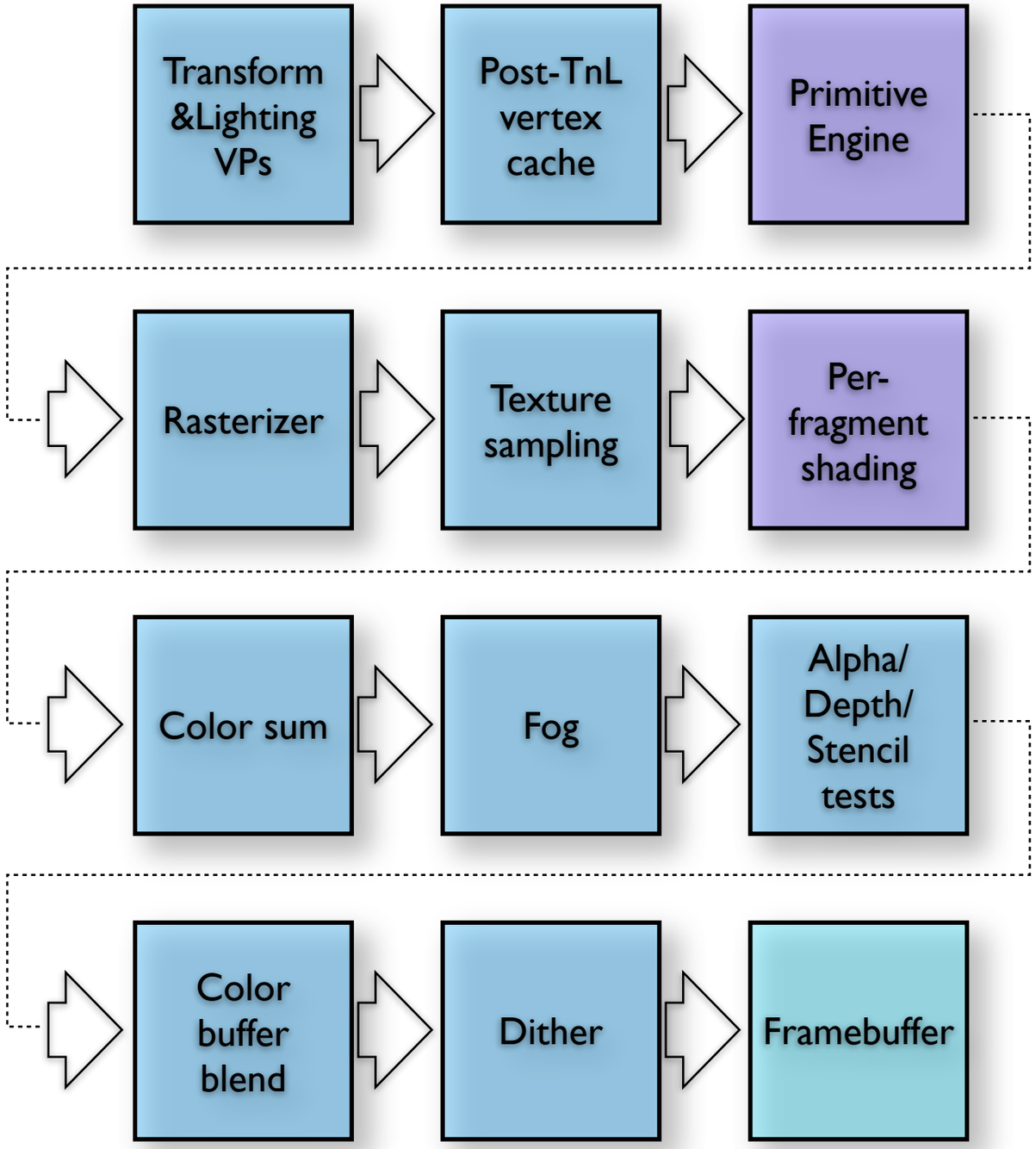
- Bring appealing shading to embedded space
- Make complex geometry processing available for embedded applications
- Sufficient performance
- Meet embedded space limitations
  - Minimize gate size, power consumption and memory traffic growth
- Heterogeneous architecture as a solution

Related work

- Partially programmable IPs [Woo et al. 2004; Donghyun Kim 2005; Imai et al. 2004; Kameyama et al. 2003] gradually replaced by programmable [IMG, NVIDIA, ARM etc] ones
- Subdivision/high-order surfaces tessellation as geometry compression
  - Specifically tailored solutions [Uesaki et al. 2004; Pedersen 2004]
  - Multipass techniques [Shiue et al. 2005; Andrews and Baker 2006]
  - Geometry shader-based ones [Loop et al. 2009]
- Real-time BRDF rendering
  - BRDF factorization into 2D functions [Heidrich and Seidel 1999]
  - Half vector-based parametrizations [Kautz and McCool 1999]
  - Factorization into combination of 1D functions [Lawrence et al. 2004; Lawrence et al. 2006]
  - Fixed HW implementation for certain shading models [Ohbuchi and Unno 2002]

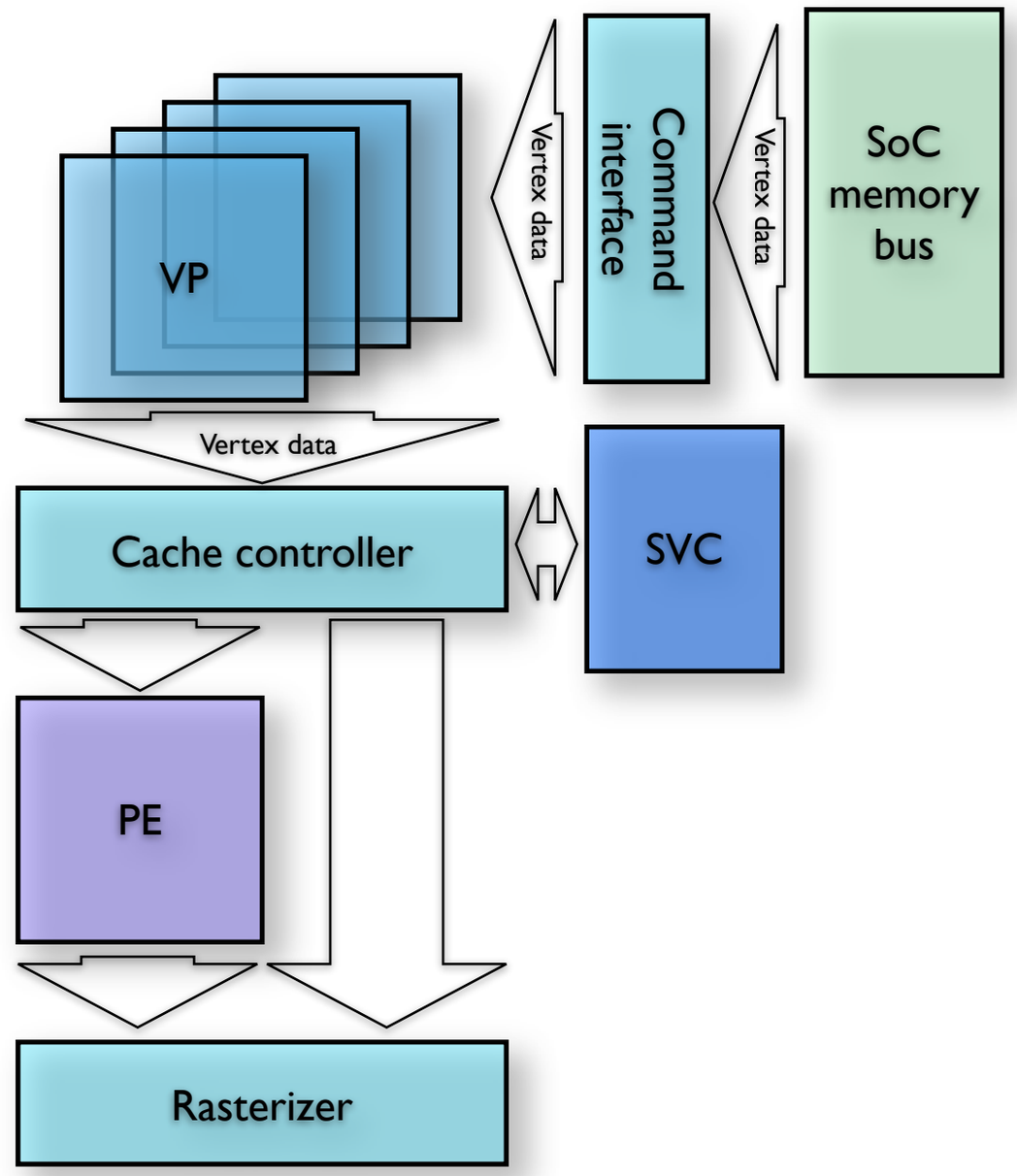
Architecture overview

- Programmable geometry processing + fixed function fragment shader
- Augmented OpenGL ES 1.X pipeline
- Primitive Engine
  - Extended VP
- Fixed-function fragment shader
  - Calculates shading based on interpolated local frame and view info
  - Consumes bump/tangent/ shadow map samples
  - Provides extra inputs to texture environment



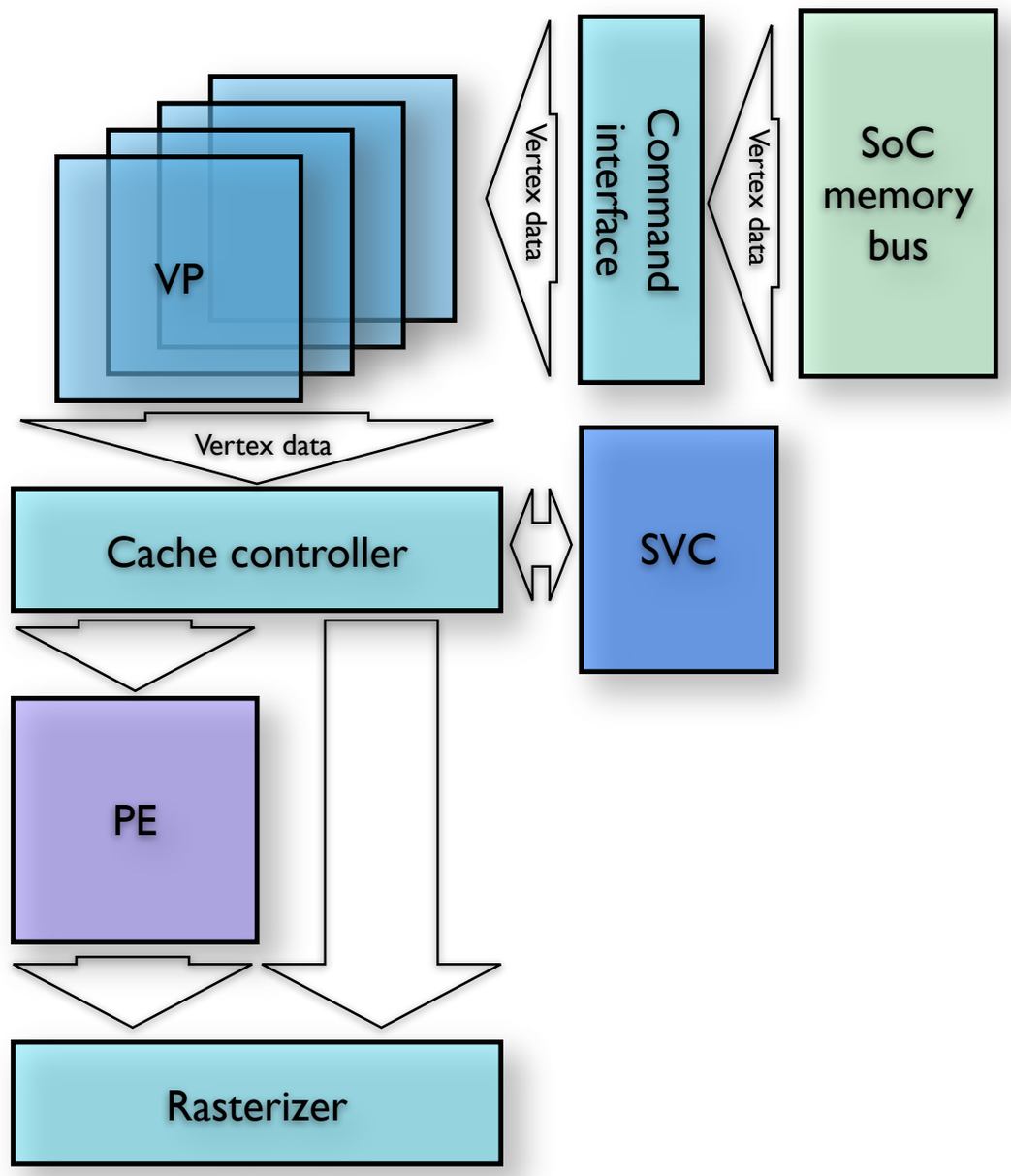
Geometry engine

- SM 3.0-level Vertex Processors
- Secondary vertex cache (SVC) + Primitive Engine (PE) combination
  - PE is a VP with programmable primitive output
- Fixed and variable size geometry primitives
  - Up to SVC size per primitive



Geometry engine

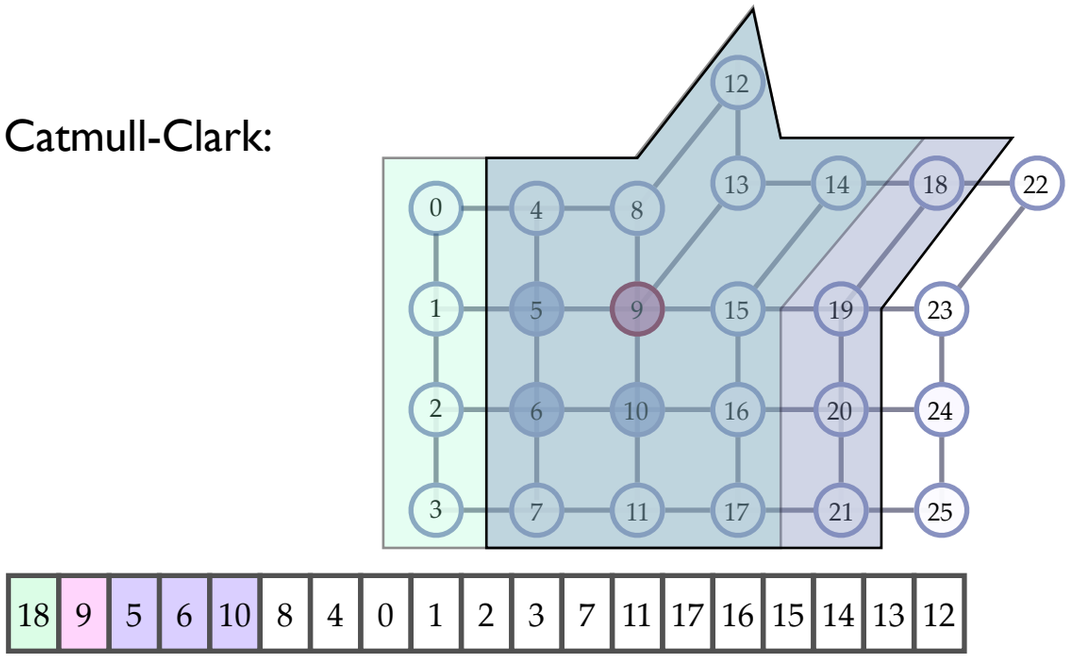
- All geometry shader`s geometry input comes from SVC
  - No need for texture access
- SVC exploits spatial coherency
  - VB traffic reduction
  - Important for multivertex primitives and complex geometry processing algorithms
- Optional reduction of internal vertex attribute traffic
  - Full set of attribs for a few initial primitive vertices
- Marginal gate size growth
  - Gate estate sharing with VPs
  - Limited modification of SVC logic



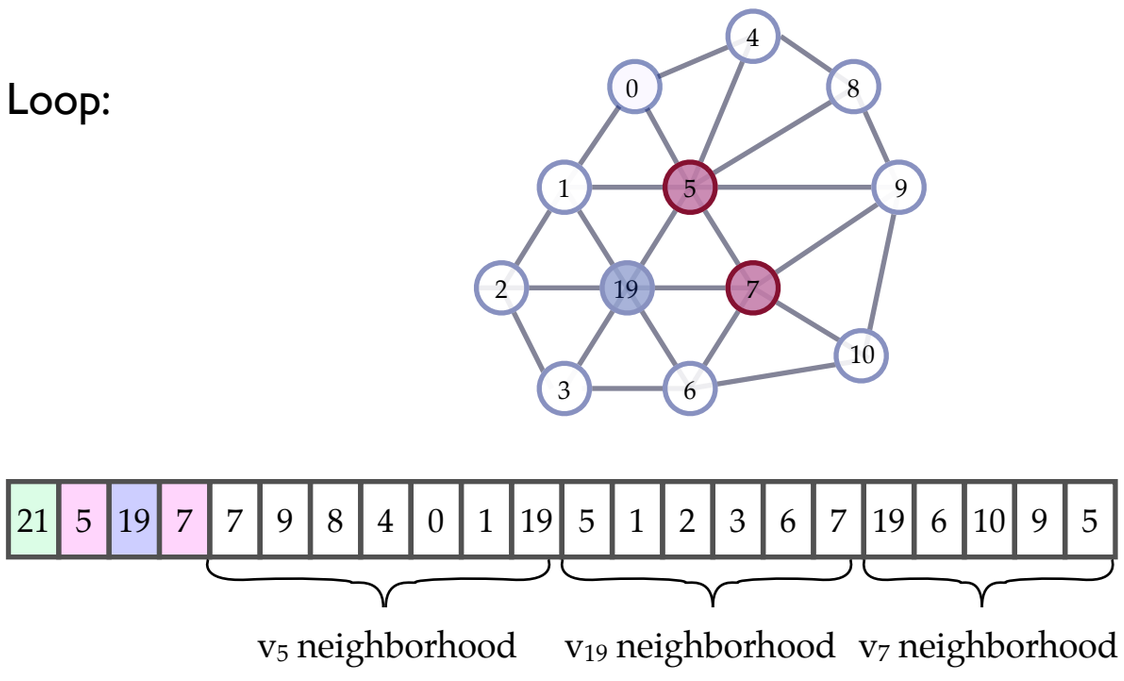
Variable-size primitives

- Primitive size prefixes vertices in index buffer
- Variable primitive size for GS
- Subdivision implementation
  - Supports varying patch size sequence naturally
  - One shader for all patches
  - No coherency breaks
  - No texture access for connectivity information
  - Subdivision patches are big and share a lot - greatly accelerated by vertex cache

Catmull-Clark:



Loop:



Fragment shader

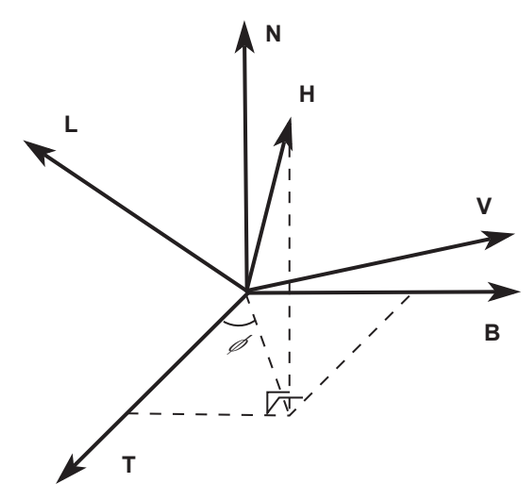
- OpenGL ES 1.X shader + per-fragment shading module
- Primary and secondary color outputs
- Combines several 1D shading functions stored in on-chip LUTs
- Configurable LUT inputs
  - $\mathbf{N}\cdot\mathbf{V}$, $\mathbf{N}\cdot\mathbf{L}$, $\mathbf{N}\cdot\mathbf{H}$, $\mathbf{V}\cdot\mathbf{H}$, $cos(\phi)$, spot
- Alpha output from LUT
  - Used for Fresnel-like reflection
- LUT output can be disabled (constant)
- Physically-based and NPR shading models
  - Multilayer reflection can be approximated as well

$$C_p = m_e + m_a s_a + \sum_{i=0}^{n-1} AS(d_0) f_i H(m_a l_{ia} + m_d l_{id}(\mathbf{L}\cdot\mathbf{N})) \quad (1)$$

$$C_{s\lambda} = \sum_{i=0}^{n-1} AS(d_0) f_i H(m_{s\lambda} D_0(d_1) G_0 + R_\lambda(d_2) D_1(d_3) G_1) l_{si\lambda} \quad (2)$$

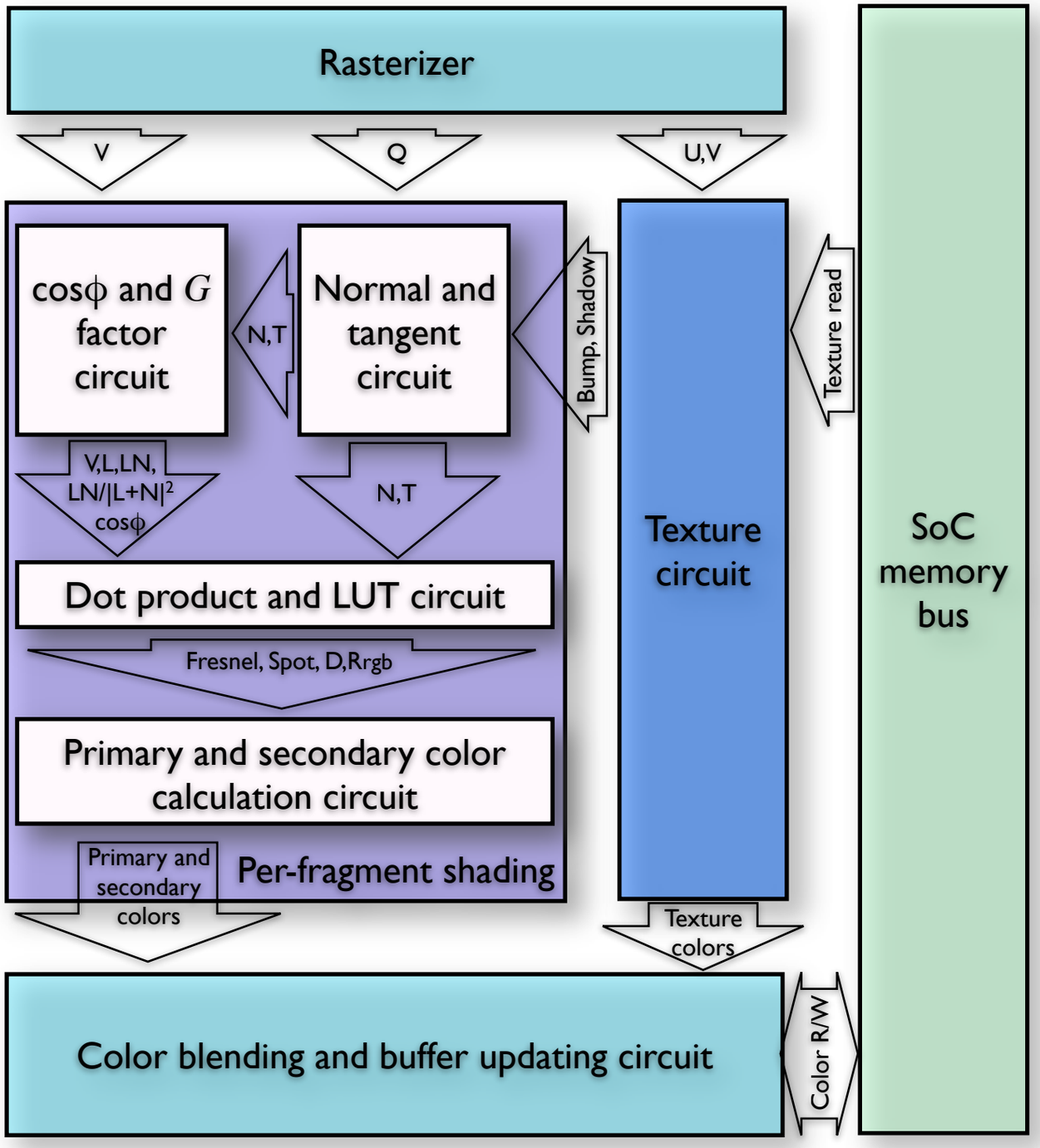
$$G_{0,1} = G' \quad or \quad 1,$$
$$G' = (\mathbf{L}\cdot\mathbf{N}) / |\mathbf{L}+\mathbf{V}|^2$$

Angle ϕ:

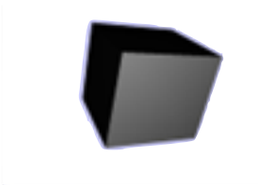


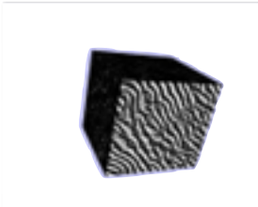
Fragment shader

- Multiple lights
- Perturbation by bump/tangent map
- Attenuation by shadow map
- Local frame reconstruction from quaternion
- A long fixed function pipeline
  - 30-50 stages
  - Aligns with texture access latency
  - Matches 3 24bit 4way SIMD units in size
- All LUTs are on-chip
  - Zero external memory access during rendering
- Fixed time per fragment
  - 1-4 clocks/fragment/light depending on configuration
  - Predictable performance



Shading performance





Shading model	Clk/frag	SM 3.0 asm steps
Phong shading model $D_0 = cos^{s}(\mathbf{N} \cdot \mathbf{L})$ $G_{0,1}=1$	1	35
Phong + bump $D_0 = cos^{s}(\mathbf{N'} \cdot \mathbf{L})$ $G_{0,1}=1$	1	38
Schlick anisotropic model $D_1 = Z(\mathbf{N} \cdot \mathbf{H}), R_\lambda=F_\lambda(\mathbf{V} \cdot \mathbf{H})$ $S=A(cos\phi), G_{0,1}=G'$	4	61
Cook-Torrance shading model $D_1 = D(\mathbf{N} \cdot \mathbf{H}), R_\lambda=F_\lambda(\mathbf{V} \cdot \mathbf{H})$ $G_{0,1}=G'$	2	48

1.X API

- Dedicated APIs in the case of 1.X library
- In spirit of 1.X API for FS
  - Light reflection environment (similar to texture environment)
  - 8 API functions
    - Per-fragment shading and LUT management
    - A lot of extra tokens
- Preconfigured geometry shaders selected according to a primitive type
  - Subdivision, silhouette, particle systems

```
glActiveLightDMP ( GL_LIGHT0_DMP ) ;
glLightEnviDMP ( GL_LIGHT_ENV_LUT_INPUT_SELECTOR_D0_DMP
               , GL_LIGHT_ENV_LN_DMP ) ;
glLightEnviDMP ( GL_LIGHT_ENV_LAYER_CONFIG_DMP
               , GL_LIGHT_ENV_LAYER_CONFIG0_DMP ) ;
glLightEnviDMP ( GL_LIGHT_ENV_GEOM_FACTOR0_DMP, GL_FALSE ) ;

GLfloat lut[512] ;
for ( j = 1 ; j < 128 ; j++ ){
     lut[j] = powf( (float)j/127.f, 30.f ) ;
     lut[j+255] = lut[j] - lut[j-1] ;
}

glMaterialLutDMP ( 2, lut ) ;
glMaterialfv ( GL_FRAGMENT_FRONT_AND_BACK_DMP
             , GL_MATERIAL_LUT_D0_DMP, 2 ) ;
```

```
GLushort indices[] = {
          14, 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13
          } ;

glDrawElements ( GL_SUBD_PRIM_DMP, 15
               , GL_UNSIGNED_SHORT, indices ) ;
```

2.0 API

- Preinstalled fragment shader object
- Exposes predefined (~200) uniforms for all parameters of fragment pipeline
- Predefined attributes for binding with VS/GS
- GL program objects are great in setting all params with a single glUseProgram call
- Other extensions to switch a set of LUTs in one call
- Minimal modifications of app/content creation chain

```
glAttachShader( progid, GL_DMP_FRAGMENT_SHADER_DMP );
glUniform1i( glGetUniformLocation( progid
                                   , "dmp_LightEnv.lutInputD0")
           , GL_LIGHT_ENV_LN_DMP );

glUniform1i( glGetUniformLocation( progid
                                   , "dmp_LightEnv.config")
           , GL_LIGHT_ENV_LAYER_CONFIG0_DMP );

glUniform1i( glGetUniformLocation( progid
           , "dmp_FragmentLightSource[0].geomFactor0"), GL_FALSE);
GLfloat lut[512] ;
for ( j = 1 ; j < 128 ; j++ ){
     lut[j] = powf( (float)j/127.f, 30.f ) ;
     lut[j+255] = lut[j] - lut[j-1] ;
}
glBindTexture(GL_LUT_TEXTURE0_DMP, lutid);
glTexImage1D( GL_LUT_TEXTURE0_DMP, 0, GL_LUMINANCEF_DMP, 512, 0
            , GL_LUMINANCEF_DMP, GL_FLOAT, lut);

glUniform1i( glGetUniformLocation( progid
           , "dmp_FragmentMaterial.samplerD0"), 0);
```

```
varying vec3 dmp_lrView;
varying vec3 dmp_lrQuat;
....
dmp_lrView     = -gl_Position.xyz;
gl_Position    = u_projection_matrix * gl_Position;
gl_TexCoord[1] = vec4(a_texcoord1.x, a_texcoord1.y, 0.0, 1.0);
gl_FrontColor  = u_material_constant_color0;
```

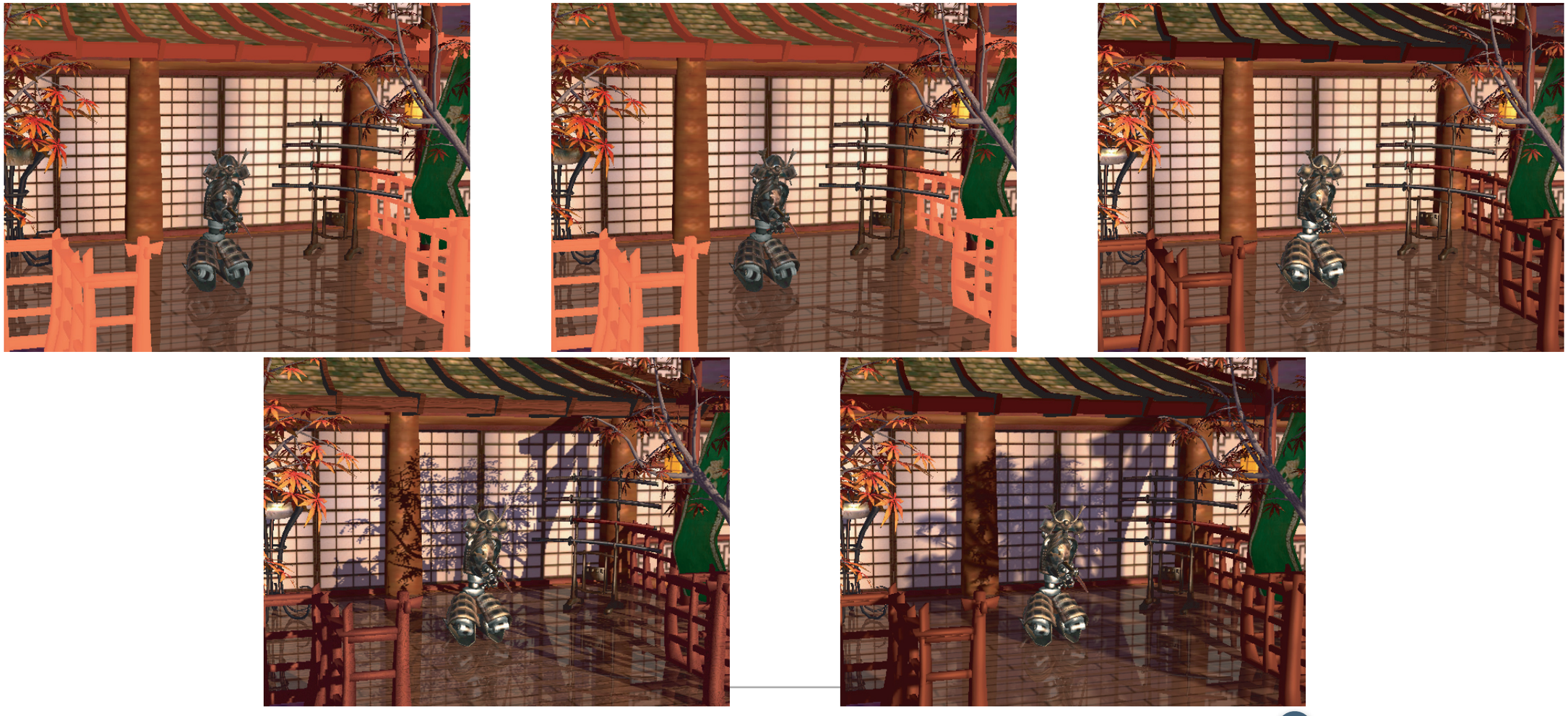
2.0 API

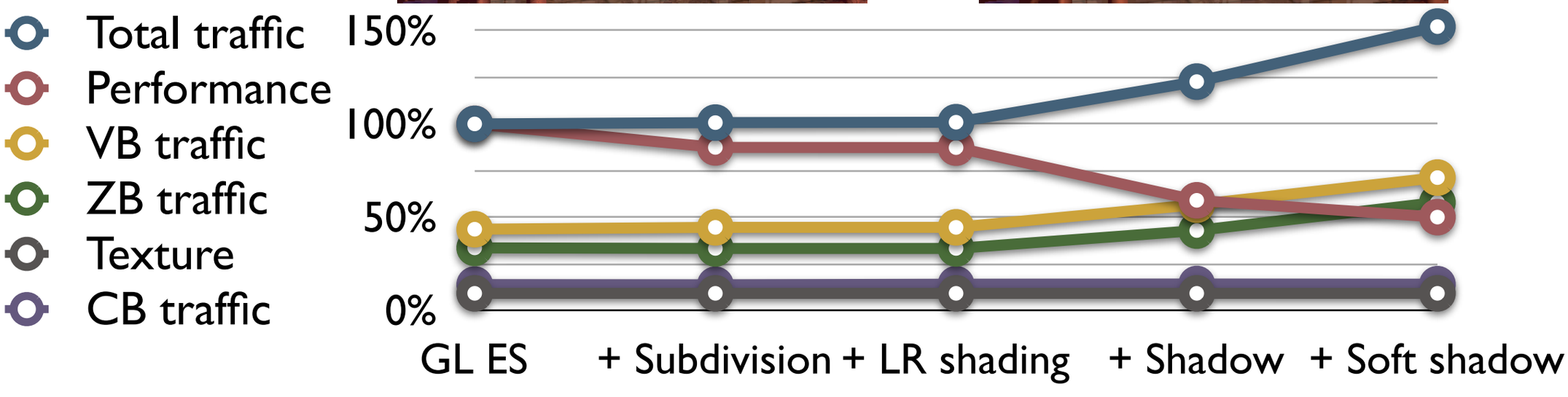
- Standard VS shader API
- GL 3.2-like GS API
- Extended primitive type for variable-size and non-standard fixed-size ones
- GLSL`s `gl_VerticesIn` is not a constant

```
GLushort indices[] = {
          14, 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13
          } ;
glUniform1f( glGetUniformLocation(progid
          , "subdivisionlevel"), 2);
glDrawElements( GL_GEOMETRY_PRIMITIVE_DMP, 15
             , GL_UNSIGNED_SHORT, indices ) ;
```

```
void main(void){
vec4 sc, se, v20 ;
float val = (gl_VerticesIn-8)/2 ;
if ( 3.0==val ) { // valence 3
    sc = gl_PositionIn[2] + gl_PositionIn[4] + gl_PositionIn[12] ;
    se = gl_PositionIn[1] + gl_PositionIn[3] +
         gl_PositionIn[gl_VerticesIn-1] ;
    e00   = gl_PositionIn[gl_VerticesIn-1] ;
    e0k04 = gl_PositionIn[3] ;
    c0k04 = gl_PositionIn[12];
} else { // 4 or more
    sc = sumc() ;
    se = sume() ;
    e00  = gl_PositionIn[13];
    e0k04    = gl_PositionIn[3] ;
    c0k04    = gl_PositionIn[gl_VerticesIn-2];
}
...
```

Profiling results



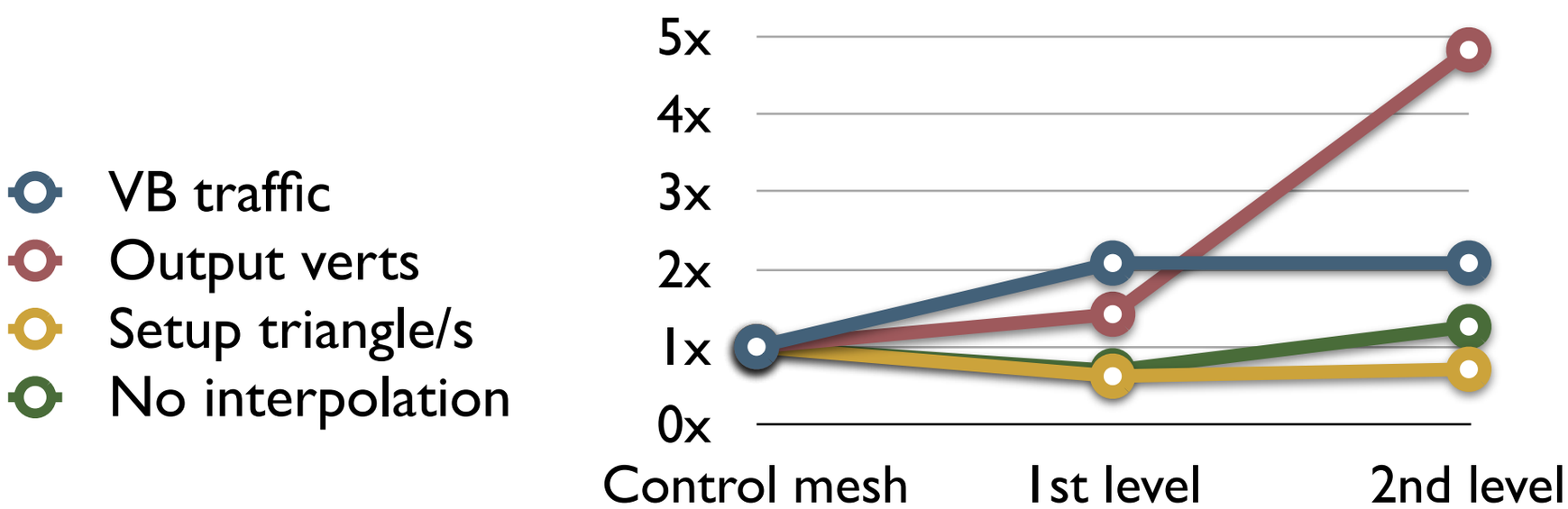


Results-subdivision

control mesh

level 1

level 2



Results-subdivision

- Lower performance than of pretessellated rendering
  - Single PE is a bottleneck for heavy shaders
- 800+ instructions in CC shader
  - One irregular vertex only
- 700+ instructions in Loop shader
  - Up to 3 irregular vertices
- ~50% of interpolation instructions
  - Explains HW tessellators in desktop accelerators
- ~2x vertex buffer traffic growth compared to control mesh rendering
  - Vertex cache exploits a great portion of spatial coherency
  - 8x less than of pretessellated mesh rendering (2 levels)
- Patch sorting causes 7-70% increase in VB traffic
  - Depending on the object and subdivision scheme
  - Sort breaks coherency as same size primitives are not necessarily neighbors
  - Loop primitive is bigger - sort impact is heavier

Conclusion

- Hybrid architecture for embedded space
  - Predictable fragment shader performance
  - Complex geometry processing capabilities
- Vertex cache-accelerated processing of fixed- and variable-size primitives
  - Reduced VB traffic due to preserved spatial coherency
  - On-chip subdivision and silhouette rendering as illustrations
- Bump/Tangent/Shadow-mapped shading at few clk/fragment
  - Support for complex shading models
  - No extra memory access due to on-chip material data
- Extended functionality exposed via both 1.X and 2.0 OpenGL ES API
  - Enables short porting times for OpenGL ES apps/content creation chains