1. Problem with current rendering algorithms

Scene → Triangles → Rasterization / Rays → Pixels
It is simulation based.
render images by simulating light interacting with triangles.

Core idea: An image is a mathematical signal, not a geometric process.
replaces simulation with direct mathematical formulas and systems.
Instead of computing light transport every frame, RFR represents the entire visible scene as a compact analytic function and evaluates it directly.

2. Concepts

a. Images as Functions
   $I(\omega) : S^2 \to R^3$
   maps every viewing direction ω to a color.
   Typical rendering algorithms compute I by physics simulation
   My goal is to approximate it using basis functions using radiance functions

3. Mathematical model

a. Basis expansion
   The entire image is represented as

$$L(\omega, t, p) = \sum_{i=1}^{N} a_i(c,t)\Phi_i(\omega)$$

   N is a small constant (e.g. 16–64)
   $\Phi_i(\omega)$ are fixed at compile time (something like an image template)
   Only coefficients a_i change at compile time
   These basis functions are global lighting patterns

4. Rendering algorithm

a. Frame evaluation

For each frame

- Compute coefficients a_i from camera and time
- Evaluate basis Φ_i for each pixel direction
- Accumulate image

No geometry, no rays, no depth buffers

5. System level flow

a. Pipe line

Scene → Radience compiler → coefficient processing + basis set → Radience processing unit → Display image

6. Radience compiler

a. The compiler replaces asset pipelines.

3D scene → Global sampling → basis projection → Coefficient evaluation →Radience program

Something similar to how a video is pre encoded instead of simulated

7. Runtime pipeline

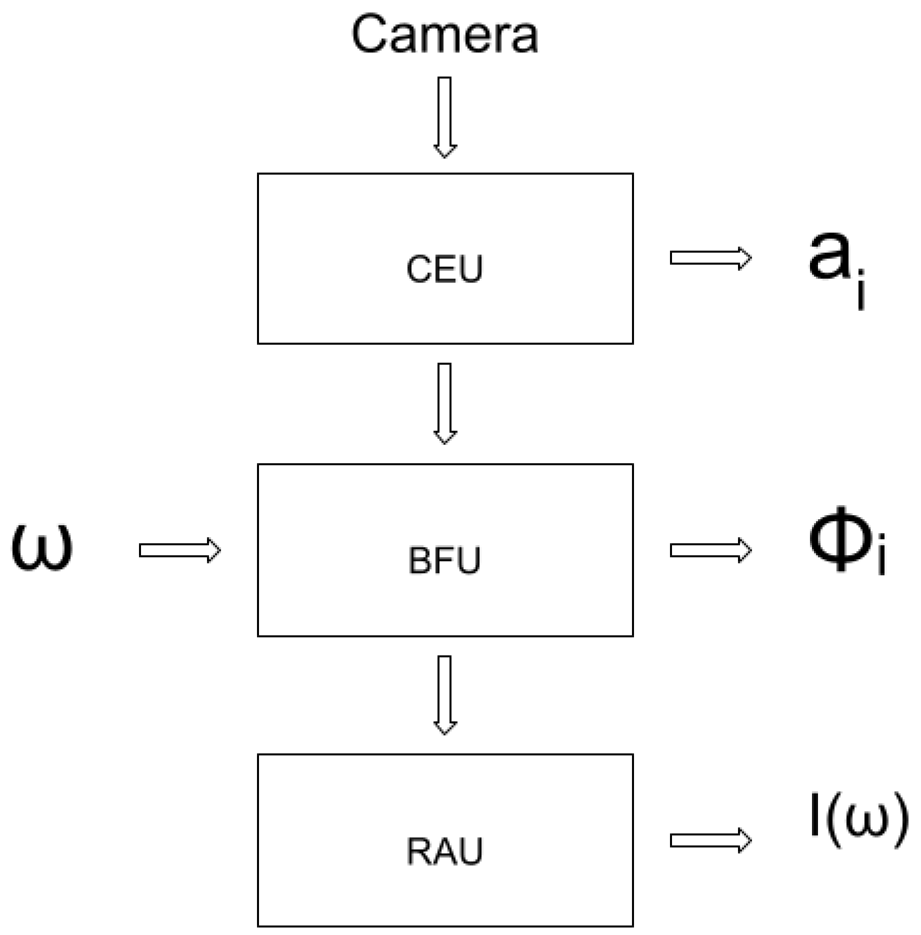
a. Per frame execution

Camera state→ coefficient evaluation unit → basis function unit → radiance accumulation unit → framebuffer

8. Radiance processing unit (RPU)

Not a gpu, more like a signal synthesiser
Each core renders a full frame
Frame rate can theoretically scale linearly with cores
1000 cores → 1000 frames per computation cycle

a. **Core structure**



9. Coefficient Evaluation Unit (CEU)

Computes:
$a_i = c_0 + c_1x + c_2y + c_3z + c_4\sin(t) + c_5\cos(t)$
Fully parallel across all basis

10. Basis Function Unit (BFU)

Evaluates spherical harmonic or learned analytical bases
Example:
$\Phi_i(\omega) = Y_{\square}^{m}(\theta, \varphi)$

11. Radiance Accumulation Unit (RAU)

Per pixel:
$I(\omega) = \Sigma\, a_i\, \Phi_i(\omega)$
Implemented as a 32 wide vector FMA + reduction tree.

12. Instruction set architecture (ISA)

Instruction	Function
CEVAL	Evaluate Coefficients
BEVAL	Evaluatte basis
VFMA	Vector multiply accumulate
REDUCE	Sum vector lanes
EMIT	Write pixels

13. Pipeline timing

Estimated:

Stage	Cycles
CEU	6
BFU	4
RAU	3
output	1

Total ~14 cycles per frame per core

14. Memory Model

No vertex buffers. No textures. No depth buffers

What is there:

storage	size
Basis RAM	KBs
Coefficient RAM	KBs

15. Parallel scaling

Each core renders independently
Performance scales with cores linearly

16. Comparison with other GPUs

Feature	GPU	RPU
Geometry	Heavy	None
Memory	TB/s	Minimal
Complexity	High	Lower
Scaling	Sublinear	Linear

17. Hybrid system

RPU handles global illumination.
GPU habdels:

- UI
- Text
- Particles

18. Implementation

a. Software prototype
Use vectorized CPU code to validate basis accuracy

b. FGPA prototype
Implement CEU + BFU + RAU pipeline

19. Limitations

Issue	Cause
Sharp Edges	Low frequency basis
Dynamic topology	Needs recompilation

20. Mathematical derivations

a. Function space representation

The image is a radiance function:

$$I(\omega) \in L^2(S^2)$$

This is a Hilbert space of soiree integrable functions on the sphere

$$I(\omega) = \Sigma \langle I, \Phi_i \rangle \Phi_i(\omega)$$

Where:

$$(f, g) = \int_{S^2} f(\omega)\, g(\omega)\, d\omega$$

Therefore, the coefficients are:

$a_i = (I, \Phi_i)$

Similar to the maths that allows the Fourier Series to represent any signal

b. Truncated expansion

Using only N terms:

$$I_N(\omega) = \sum_{1}^{N} a_i\, \Phi_i(\omega)$$

The approximation error is:

$$\| I - I_N \|^2 = \sum_{i>N} | a_I |^2$$

The energy outside the first N basis defines the visual error

21. Error Bounds and Quality Analysis

a. Convergence Behavior

Natural images exhibit spectral decay:

$| a_l | \approx O(i^{-p})$ $\quad p \approx 2$ to 4

Error decreases rapidly:

$|| I - I_N || \approx O(N^{-(p-1)})$

This ensures high quality with small N

b. Visual artifact types

Artifact	Cause
Ringing	Gibbs Phenomenon
Blur	Missing high frequency basis
Ghosting	Underfitted coefficients

22. Real Basis function definitions

a. Spherical harmonics

$\Phi_{l,m}(\theta,\varphi) = N_{l,m} P_l^m(\cos\theta)e^{im\varphi}$

$N_{l,m}$ is a normalization constant

$P_l^m(\cos\theta)$ is the associated Legendre polynomial evaluated at $\cos\theta$

$e^{im\varphi}$ is the complex exponential function representing the azimuthal dependence

l is the band

m is the order

b. Zonal Axis

Axis aligned basis

$\Phi_l(\theta) = P_l(\cos\theta)$

Lower cost, good for sky and global lighting

c. Learned analytic Basis

Train neural network to produce orthogonal functions, then approximate as polynomials for hardware evaluation

23. Hybrid GPU integration

a. Task partitioning

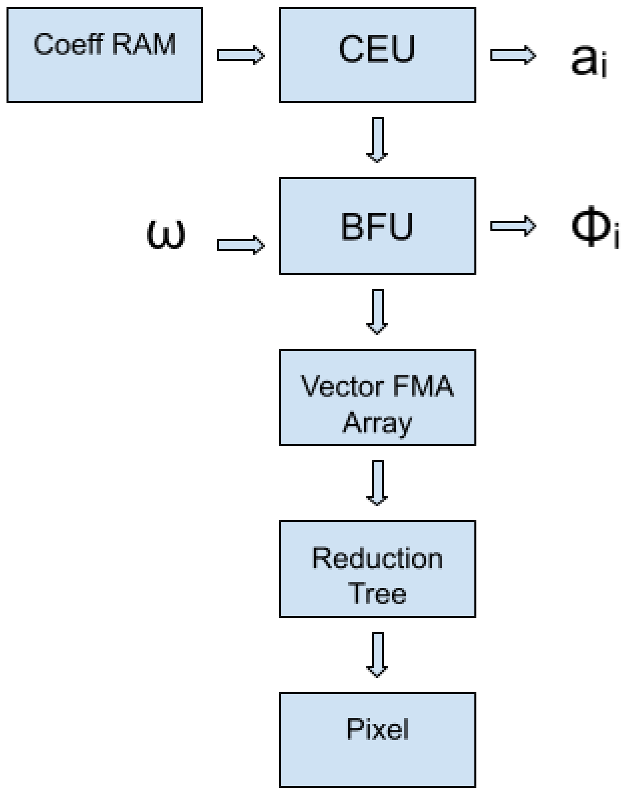
Subsystem	processor
Global lighting	RPU
Geometry edges	GPU
UI/ Text	GPU
Particles	GPU

b. Composite pipeline
RPU → GPU raster pass → final display

RPU acts as a background radiance synthesizer

24. Detailed Microarchitecture

a. Core datapath



b. CEU pipeline
Regs → FMA → FMA → FMA → TRIG → Output

Fully pipelined, one coefficient per cycle

c. BFU pipeline
ω → sin/cos → polynomial → normalize

d. RAU pipeline
32-wide FMA → 16-wide → 8 → 4 → 2 → 1

Tree reduction for deterministic latency

25. Numerical example: toy Scene encoding

a. Scene definition

Consider a simple scene

- Blue sky
- Green ground
- Bright sun (at $\theta=30°$, $\varphi=0°$)

True radiance function

$I(\omega) = 0.5 + 0.3\, Y_1^0(\omega) + 1.2\, Y_2^1(\omega)$

b. Coefficient computation

Using orthonormality:

$$a_i = \int I(\omega)\, \Phi_i(\omega)\, d\omega$$

Results:

Basis	a_i
Y_0^0	0.5
Y_1^0	0.3
Y_2^1	1.2

All other coefficients approximately 0

c. Runtime evaluation

Per pixel:

$$I(\omega) \approx 0.5\, Y_0^0 + 0.3\, Y_1^0(\omega) + 1.2\, Y_2^1(\omega)$$

Only 3 FMAs produce the image

26. Radience compiler pseudocode

a. Global sampling

```
for each probe $\omega_j$
    $I_j$ = offline_path_trace(scene, $\omega_j$)
```

b. Basis projection

```
for each basis $\Phi_i$
    $a_i = \sum_{j} I_j \cdot \Phi_i(\omega_j) \cdot w_j$
```

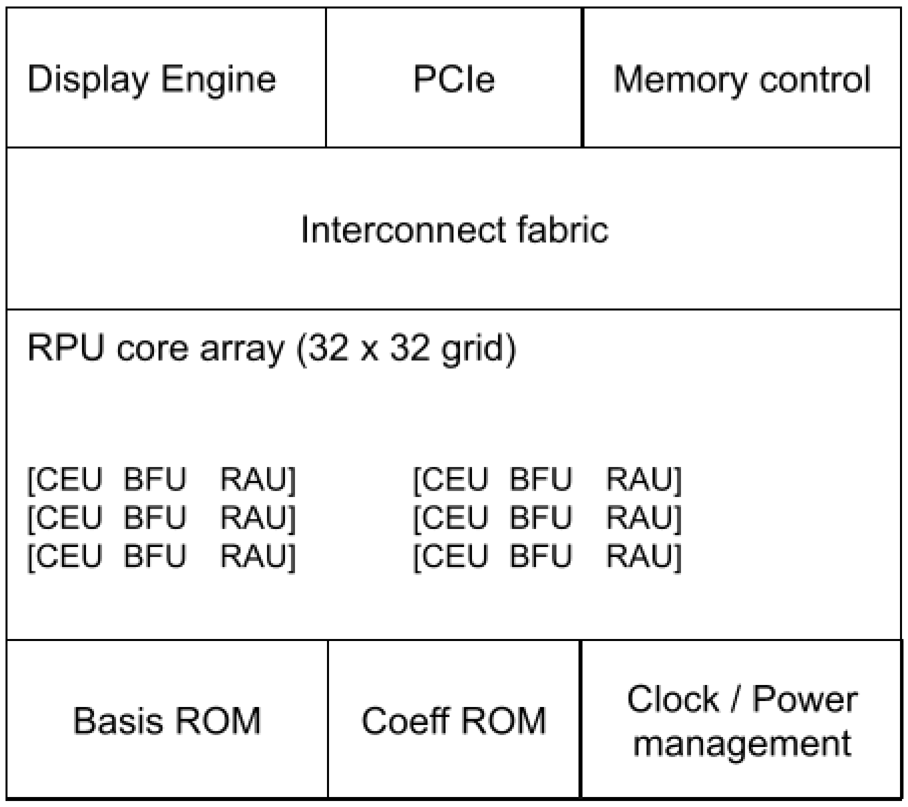
c. Coefficient regression

```
Solve min $|| a_l(cam,t) - f_i(cam,t) ||^2$
Store polynomial coefficients
```

d. Code estimation

```
Emit CEVAL instructions
Emit BEVAL instructions
Emit VFMA tree
```

27. Silicon floorplan sketch



Regular layout enables for dense packing

28. Performance comparison

a. Asymptotic Complexity

System	Complexity
Raster	O(P)
Ray Tracing	O(P log G)
RFR	O(1)

P = pixels
G = geometry

b. Energy efficiency

Metric	GPU	RPU
Memory BW	TB/s	~0
Power	300W	<50W

Based entirely on intuition and estimated guess

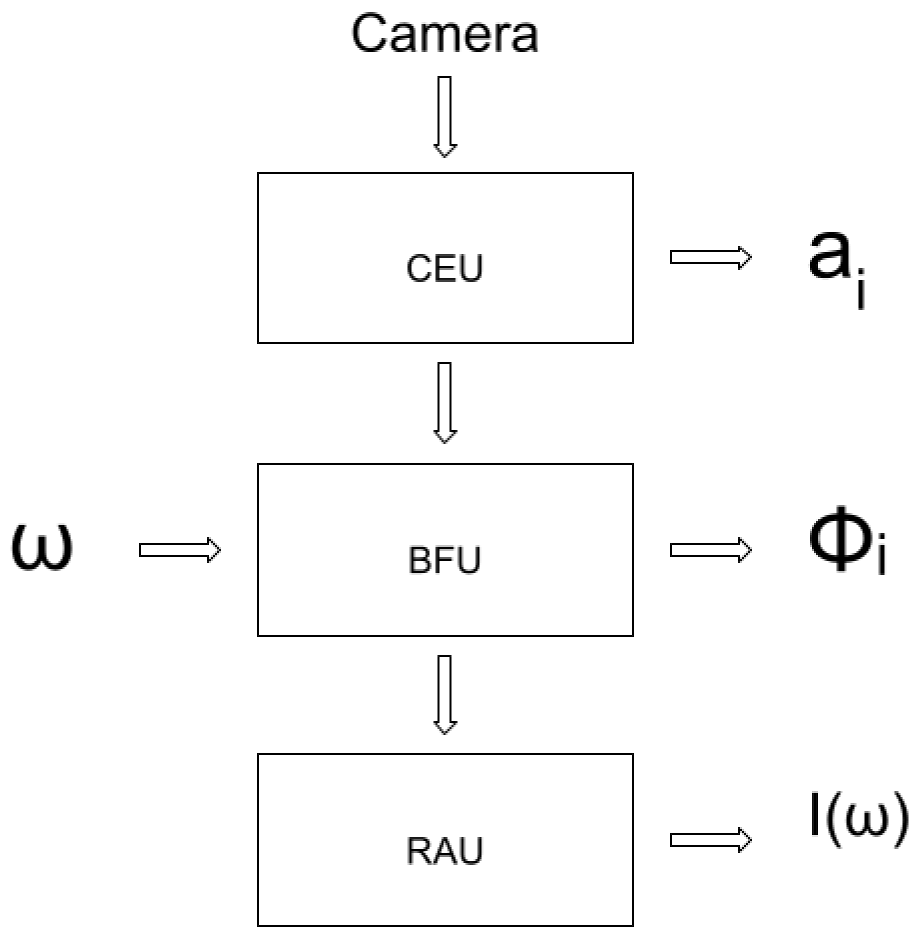
29. Diagrams

a. Classical vs RFR pipeline Diagram

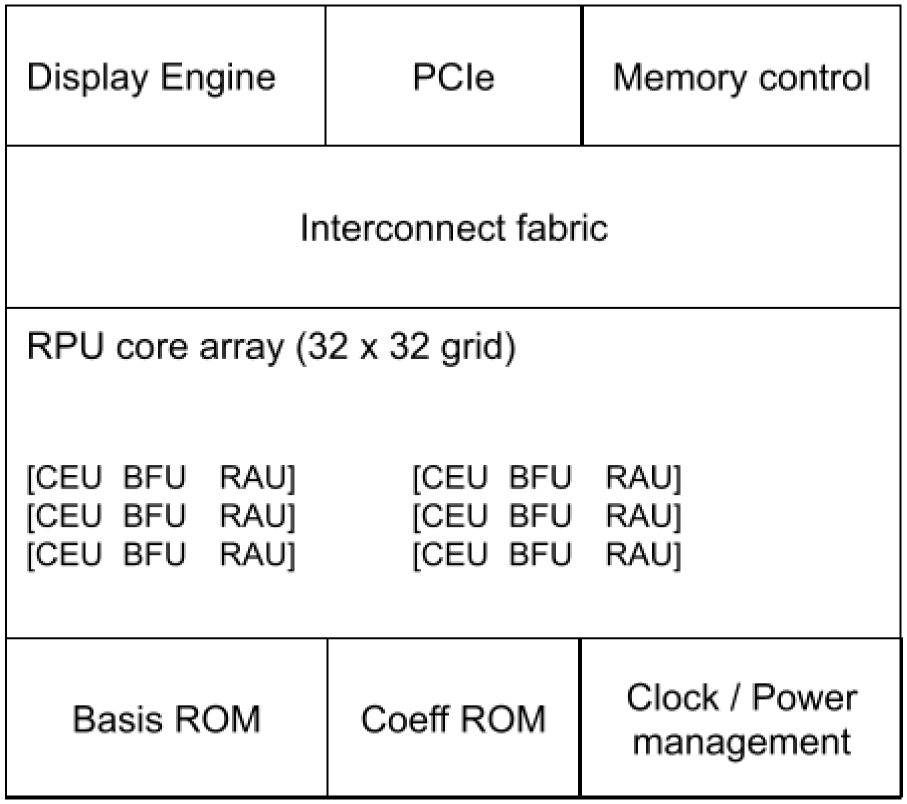
   i. Classical GPU:
      Scene → Triangles → Rasterization → shading → pixels

   ii. RFR:
      Scene → Radience compiler → coefficients + basis → RPU → pixels

b. RPU core block



c. Chip diagram



d. Compiler Flow :
3D scene → Global sampling → basis projection → Coefficient regression →Radience Program

30. Stress testing (inaccurate values for purpose of example)

a. High frequency geometr
   Problem: sharp edges require many basis terms
   Effect

N	Result
16	Blurry edges
32	Acceptable
128	Near perfect

   Mitigation: Hybrid GPU edge pass

b. Rapid topology changes
   Explosions or destruction change radiance function suddenly
   Issue: coefficients become invalid
   Solutions:
   - Recompiler coefficients
   - Or temporarily fall back to raster GPU

c. Specular caustics
   Highly concentrated light created high frequency lobes
   Requires specialized basis functions (wavelet lobes)

d. Dynamic lighting
   Fast changing lights modify coefficients only
   No geometry cost increase
   This is a major advantage over raster and RT

e. Worst case scene
   Random noise scene
   Energy spectrum is flat → slow convergence
   This defines the theoretical limit of RFR

31. Conclusion

- Radiance function rendering is not a faster rasterizer
- It is a new computational model for graphics
- By converting rendering from simulation to synthesis, it achieves constant time image generation and perfect parallel scaling
- RFR establishes a new class of processors, Radiance Evaluation Units
- Radience evaluation converts image synthesis into signal evaluation
- It is theoretically scalable endlessly
- Represents O(1) graphics architecture