Algorithms and techniques for virtual camera control

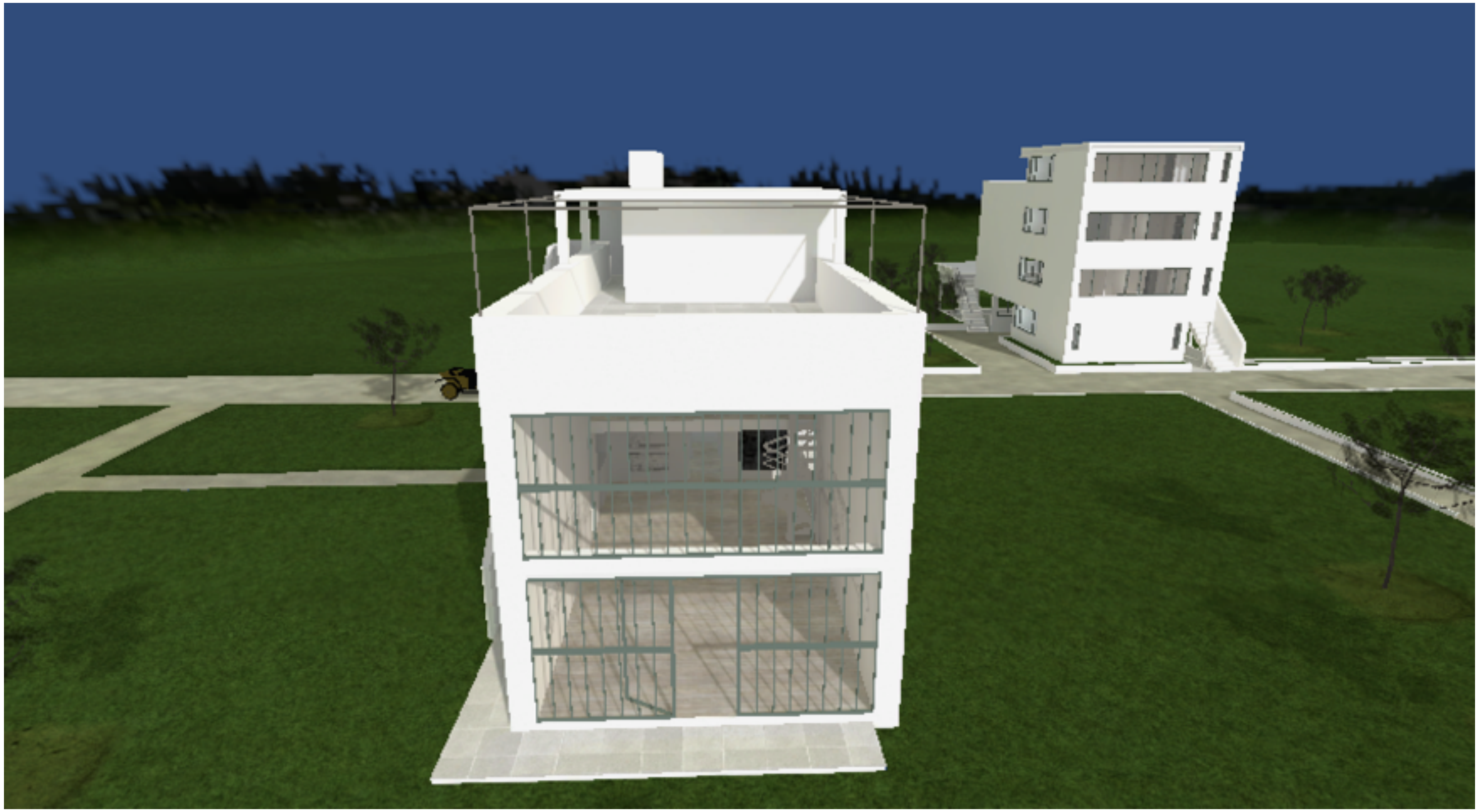
Session 4: Viewpoint Computation

M. Christie, Univ. Rennes 1
C. Lino, Univ. Rennes 1
R. Ranon, Univ. Udine

Viewpoint Computation

- given:
  - a camera model (e.g., position - orientation - FOV), and a domain $D \subset \mathbb{R}^7$ of allowed camera parameters
  - requirements about the visual composition of targets in the computed image
- compute a value for each camera parameter to (best) satisfy the requirements

Example



requirements: houses 1 and 2 completely visible, seen from the front; houses area on screen each about 10%

Approaches to VC

- **algebraic**: when we can establish an algebraic relation between requirements and camera parameters
  - works only in very limited situations
- in all other cases, we can use:
  - **constraint-based** approaches: express requirements as constraints over D, find camera parameters $\boldsymbol{c}$ that satisfy constraints, or fail
  - **optimisation-based** approaches: express requirements as a satisfaction function $F: D \rightarrow [0,1]$, find camera parameters $\boldsymbol{c}$ that maximise F

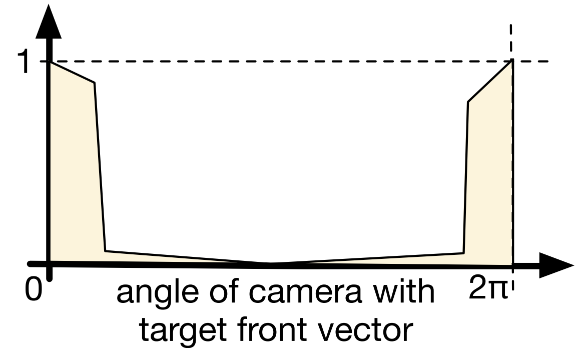
Visual Composition Requirements

- we consider the following types of visual composition requirements:
  - **size** (width, height, area) of targets on screen
  - **visibility** of targets on screen
  - **angle** of targets with camera
- we need to:
  - model each type of requirement as a satisfaction function $f:D \rightarrow [0,1]$
  - model the satisfaction function of a virtual camera ***c*** as some composition of the requirement satisfaction functions, i.e. $F(f_1,f_2,\ldots,f_n):D \rightarrow [0,1]$

Modeling requirement functions

- a requirement has a type (size, visibility,…) and a desired value
  - the "type" part computes the value *v* of a visual feature (size, visibility, angle) of a target *t* for a given camera, i.e. $f_{type}(\boldsymbol{c}): D \rightarrow V$ where V is the set of possible values of the visual feature
  - the "desired value" part computes a satisfaction value from the value of the visual feature, i.e. $f_{desired}: V \rightarrow [0,1]$ and can be e.g. modelled as a linear spline

$$f(\boldsymbol{c}) = f_{desired}(f_{type}(\boldsymbol{c}))$$

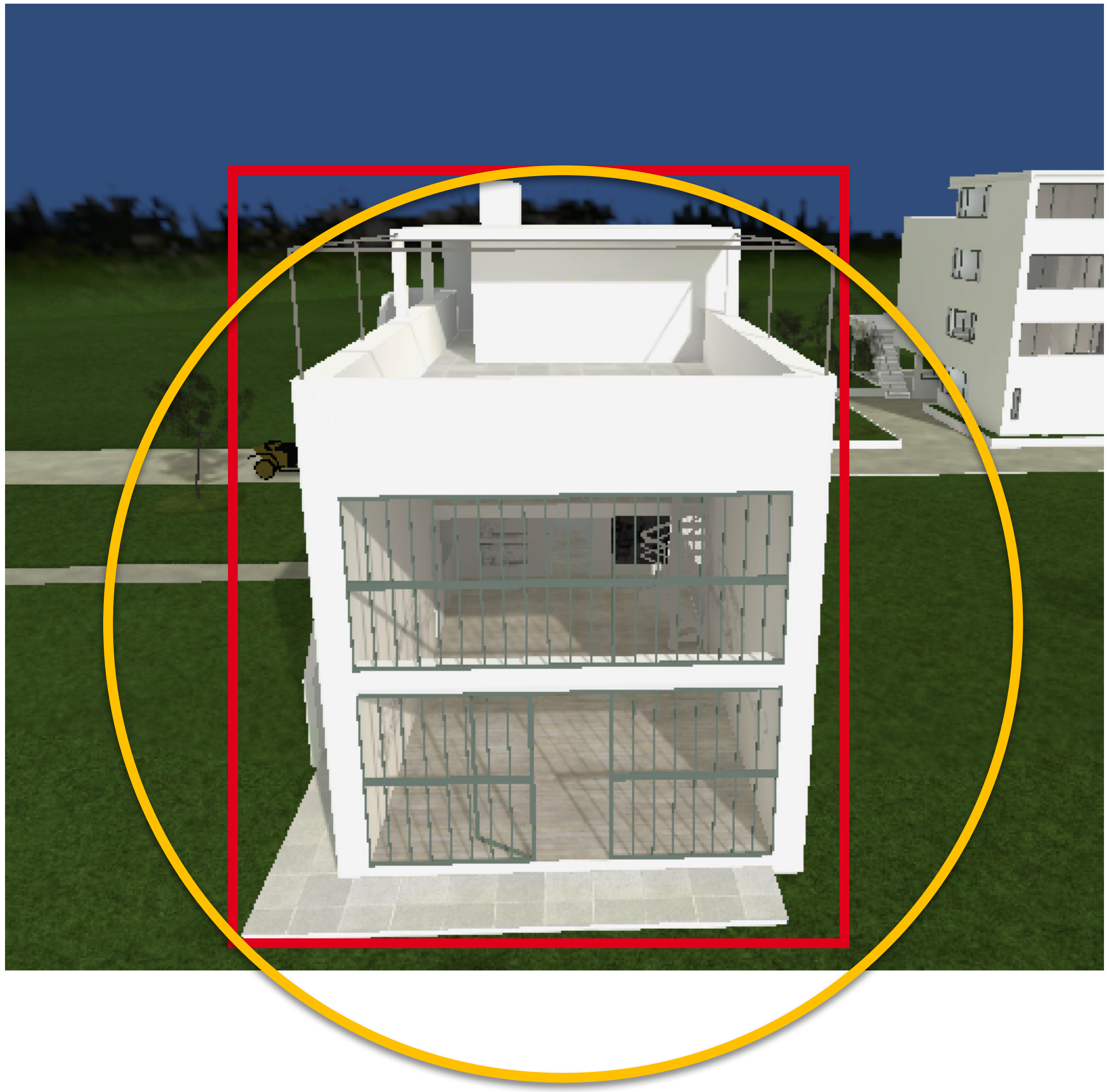


example $f_{desired}$

- optimisation approaches typically need to sample a considerable number of points in D to find a good solution: therefore, computing f_{type} must keep into account accuracy vs cost
  - compromise might depend on specific application demands

Measuring Size (area)

- mesh vs bounding volume
  - what about objects with holes
- rendering (and counting pixels)
- geometrical methods
  - bounding sphere
  - bounding box

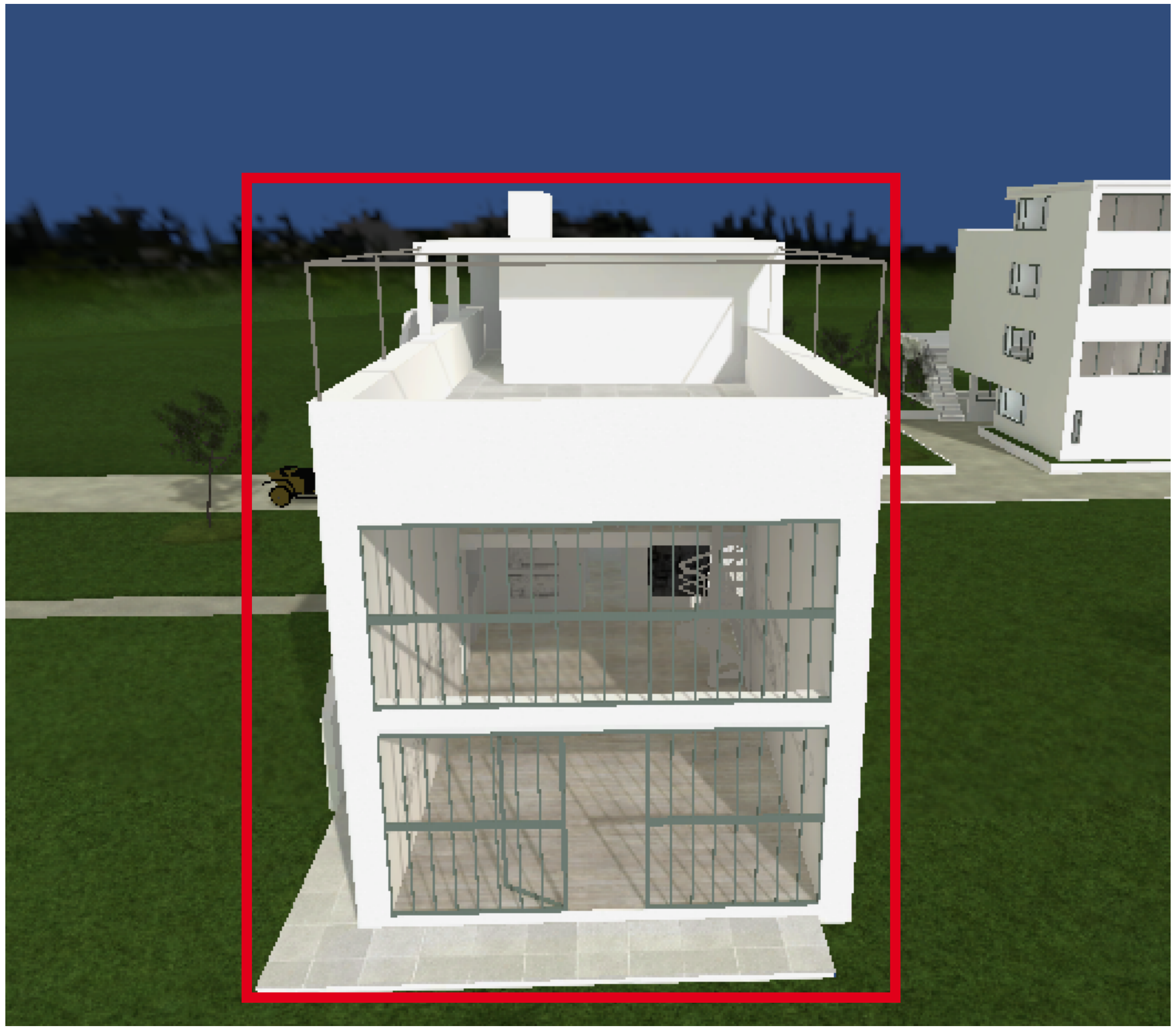


rendering at 1000x750	rendering at 500x375	rendering at 80x60	rendering at 40x30	geometric evaluation
261.87	**63.47**	**9.7**	**8.44**	**0.005**

mean evaluation cost (milliseconds) in a scene after 1000 evaluations from random cameras

Measuring Visibility

- mesh vs bounding volume
  - objects with holes
- rendering (and counting pixels)
- geometrical methods
  - ray casting

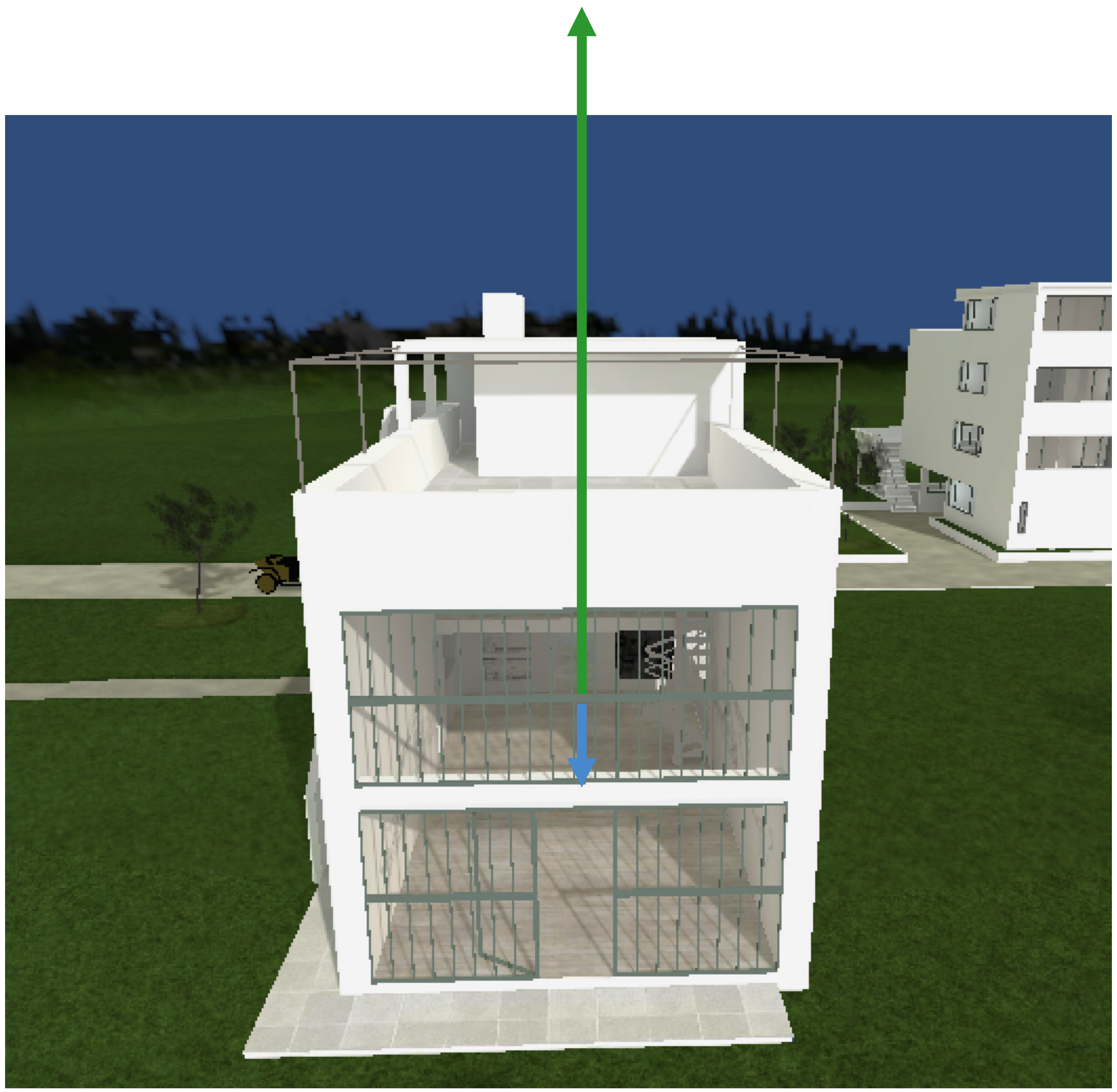


rendering at 1000x750	rendering at 500x375	rendering at 80x60	rendering at 40x30	geometric evaluation
261.68	**63.1**	**9.61**	**8.96**	**0.1**

mean evaluation cost (milliseconds) in a scene after 1000 evaluations from random cameras

Angle

- the angle between a target-specific vector ***u*** and the vector from *t* to the camera
  - typically, ***u*** can be the *forward* vector of the target (horizontal angle between camera and target), or its *up* vector (vertical angle between camera and target)
  - other choices are possible, e.g., for a character, ***u*** could be the direction of the head



Computing Satisfaction

- typical solution is a weighted sum of individual *f*

$$F(c) = \sum_i w_i f_i(c)$$

  - corresponds to logical AND of all requirements
  - weights allow one to set requirements importance, but are not easy to manage

- can use also other logical operators (e.g. OR is max)

Solving VC

- due to complexity of the objective function and non-continuity (e.g., think visibility), black-box optimisation approaches are preferable
- use of random values (stochastic optimisation) to escape local minima
- population-based approaches have the additional advantage that poor initialisation can be corrected
- Particle-Swarm Optimisation (PSO) has all these features and, moreover, it is known for fast convergence
  - used by several authors, e.g. [Burelli et al. 2008, Abdullah et al. 2011, Ranon and Urli 2014]

PSO for VC

- idea: a swarm of cameras wanders through *D* in search of the optimum (i.e. the parameters ***c*** that maximise *F*)
- at each step, we move a camera and evaluate *F* on it
- we always record:
  - the best visited position in D for each camera (***p***)
  - the global best visited position $\boldsymbol{p_g}$
- the equations for moving a camera from its position in *D* $\boldsymbol{x}^{n-1}$ to a new position $\boldsymbol{x}^{n}$ are:

$$\mathbf{v}_i^n = w^{n-1}\mathbf{v}_i^{n-1} + c_1 r_1 \left(\mathbf{p}_i^{n-1} - \mathbf{x}_i^{n-1}\right) + c_2 r_2 \left(\mathbf{p}_g^{n-1} - \mathbf{x}_i^{n-1}\right)$$

$$\mathbf{x}_i^n = \mathbf{x}_i^{n-1} + \mathbf{v}_i^n \qquad i = 1, 2, \ldots, N$$

w, c_1 and c_2 are PSO-specific parameters; r_1 and r_2 are random numbers in [0,1]

PSO for VC

```
initialize n random cameras in array CAMERAS
i=0;
while (there is still time left) {
    move CAMERAS[i];
    evaluate F(CAMERAS[i]);
    compute new local and global optima;
    i = (i+1) mod n;
}
optimum = CAMERAS[g];
```

DEMO

Improving PSO

- unlucky random initialisation coupled with little available time (e.g. few milliseconds) and/or large search space can make PSO fail
- current methods to tackle this issue are:
  - “smart” initialisation
  - lazy *F* evaluation
  - PSO parameters tuning

Smart Initialisation

- size and angle requirements are very common in VC problems
- it is quite easy to initialise a camera such that it roughly satisfies a size or angle requirement (or both)
- e.g., for size, we can compute a roughly optimal distance to a target by the formula

$$\text{distance} = \frac{\text{target size}}{\text{target's projection size}} \cdot \frac{1}{\tan(\gamma/2)}$$

where target size and projection size are easily computed by using a bounding sphere, assumed centered on the screen

- if a problem has *k* targets, we can distribute cameras among them, and initialise each camera around optimal values for the assigned target

Lazy F evaluation

- the evaluation of *F* can be terminated as soon as we know that we cannot improve on the camera local best value (*lazy evaluation*)
  - the computed value would have no effect on camera movement
- we can then order the requirements by cost of evaluation (angle, size, visibility) so that we avoid computing unnecessary (and costly) requirements
- other strategies are possible, e.g. combine lazy evaluation with computing first the projection of bounding box of all targets, and then set the satisfaction of any requirement for the same target, if the projection is off-screen, to zero

PSO parameters tuning

- the values of n (the number of cameras in the swarm), c_1, c_2, and w can greatly influence the behaviour of PSO
- given a set of scenes and VC problems, and a set of possible PSO parameter values, one can run all possible combinations, and then use statistical methods to derive optimal PSO parameter values
- in our experience, derived parameters are quite good for all similar settings

PSO parameters tuning

Scene	Triangles	Objects	Scene AABB
city	474083	324	300 x 100 x 300 (vol: 9 x 10^6)
house	324182	50	120 x 23 x 100 (vol: 276 x 10^3)
rooms	110474	240	13.9 x 3.0 x 21.8 (vol: 909.06)

5 problems per scene

Name	Meaning	Values
N	number of particles in PSO	[20, 30, 40, 60, 80, 100, 130, 160, 200, 240, 290, 340, 380]
r_part	fraction of randomly initialized particles	[0.0, 0.33, 0.66, 1.0]
c_1	PSO cognitive parameter	[0.0, 0.5, 1.0, 1.5, 2.0, 2.5]
c_2	PSO social parameter	[0.5, 1.0, 1.5, 2.0, 2.5]
ω_{init}	PSO initial inertia weight	[0.5, 1.0, 1.5, 2.0]
ω_{end}	PSO final inertia weight	[0.0, 0.5, 1.0]

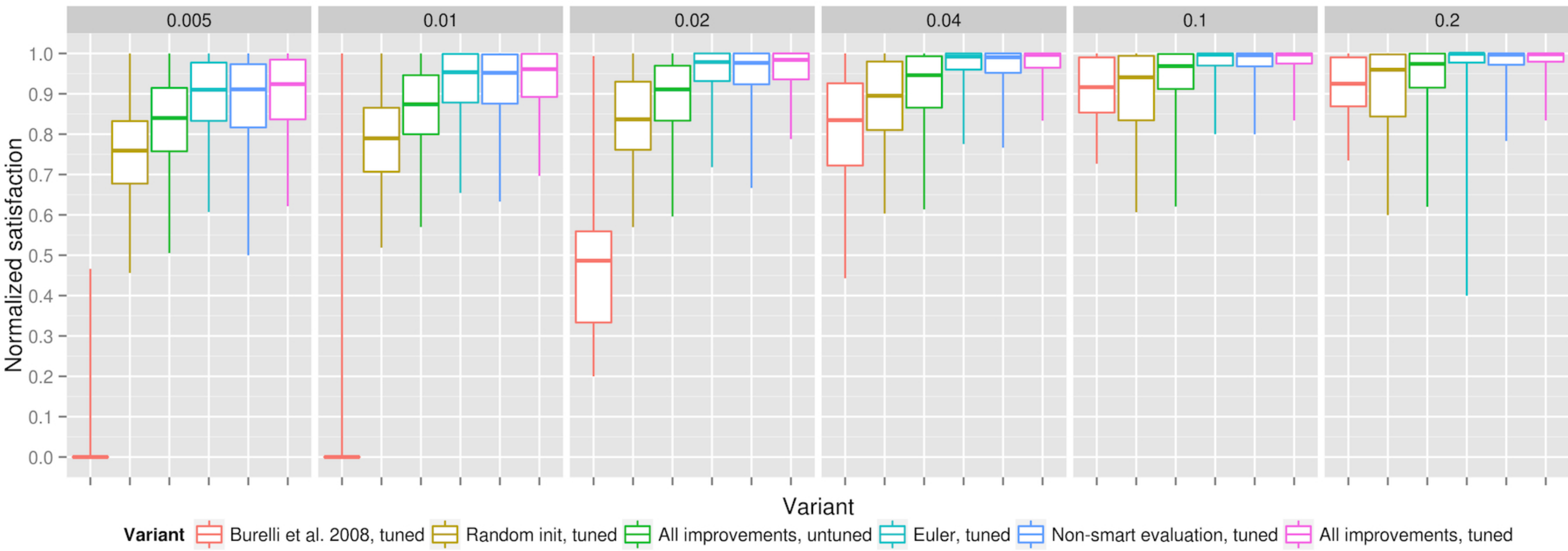
18720 combinations

20 runs per scene, problem, combination = 5’148’000 runs

considering 6 time budgets for PSO: 5, 10, 20, 40, 100, 200 milliseconds

PSO parameters tuning

T (ms)	How many	Best $(N, r\_part, c_1, c_2, \omega_{init}, \omega_{end})$	Restriction on parameters values	Median evaluations full	partial
5	102	20, 0, 2, 1.5, 1.5, 0	$N \leq 30$, $r_{part} = 0$, $c_2 \geq 1$, $\omega_{end} \leq 0.5$	60	13
10	167	30, 0, 2.5, 1.5, 0.5, 0	$N \leq 40$, $r_{part} \leq 0.33$, $c_2 \geq 1$, $\omega_{end} \leq 0.5$	108	43
20	117	30, 0, 2, 2, 0.5, 0	$N \leq 60$, $r_{part} \leq 0.33$, $c_2 \geq 1$, $\omega_{init} \leq 1.5$, $\omega_{end} \leq 0.5$	176	135
40	144	30, 0, 1.5, 2, 0.5, 0.5	$N \leq 60$, $r_{part} \leq 0.66$, $c_2 \geq 1$, $\omega_{init} \leq 1$, $\omega_{end} \leq 0.5$	373	294
100	173	130, 0, 2, 2, 0.5, 0	$r_{part} \leq 0.66$, $\omega_{init} \leq 1$, $\omega_{end} \leq 0.5$	707	625
200	218	200, 0, 1.5, 2.5, 0.5, 0.5	$c_2 \geq 1$, $\omega_{init} \leq 1$, $\omega_{end} \leq 0.5$	1204	1206



Conclusions

- VC cost is comparable to frame rendering; however, it can be spread among a few successive frames

- all techniques presented in this part, and more, are implemented in the C# Unity Library available at https://github.com/robertoranon/Unity-ViewpointComputation
  - quite easy to port to other engines (UE4 port is under way)
  - easily implement your own properties, evaluation methods, solver

References

- [Blinn, 1988] BLINN J.: Where am I? what am I looking at? IEEE Computer Graphics and Applications (July 1988), 76–81. 11, 20, 21
- [Olivier et al. 1999] P. Olivier, N. Halper, J. H. Pickering, and P. Luna, "Visual Com- position as Optimisation," in Artificial Intelligence and Simulation of Behaviour, 1999, pp. 22–30
- [Bares et al. 2000] W. H. Bares, S. McDermott, C. Boudreaux, and S. Thainimit, "Virtual 3D camera composition from frame constraints," in *Proceedings of the eighth ACM international conference on Multimedia*. ACM Press, 2000, pp. 177–186
- [Christie and Normand 2005] M. Christie and J.-M. Normand, "A semantic space partitioning approach to virtual camera composition," Computer Graphics Fo- rum, vol. 24, no. 3, pp. 247–256, 2005
- [Burelli et al 2008] P. Burelli, L. Di Gaspero, A. Ermetici, and R. Ranon, "Virtual Camera Composition with Particle Swarm Optimization," in SG '08: Proceedings of the 8th International Symposium on Smart Graphics, vol. Lecture No, no. 5166. Springer-Verlag, 2008, pp. 130–141.
- [Ranon and Urli 2014] Ranon R., Urli T., Improving the Efficiency of Viewpoint Composition, IEEE Transactions on Visualization and Computer Graphics, 20(5), May 2014, pp. 795-807.
- [Abdullah et al 2011] Rafid Abdullah, Marc Christie, Guy Schofield, Christophe Lino, Patrick Olivier. Advanced Composition in Virtual Camera Control. Smart Graphics, Aug 2011, Bremen, Germany. 6815, pp.13-24, 2011, Lecture Notes in Computer Science.
- [Bares 2006] W. H. Bares, "A Photographic Composition Assistant for Intelli- gent Virtual 3D Camera Systems," in SG '06: Proceedings of the 6th International Symposium on Smart Graphics, ser. Lecture Notes in Computer Science, vol. 4073. Springer-Verlag, 2006, pp. 172–183
- [Schmalstieg and Tobler, 1999] D. Schmalstieg and R. F. Tobler, "Real-time bounding box area computation," Tech. Rep. TR-186-2-99-05, Jan. 1999.
- [Lino, 2015] Christophe Lino. Toward More Effective Viewpoint Computation Tools. Eurographics Workshop on Intelligent Cinematography and Editing, May 2015