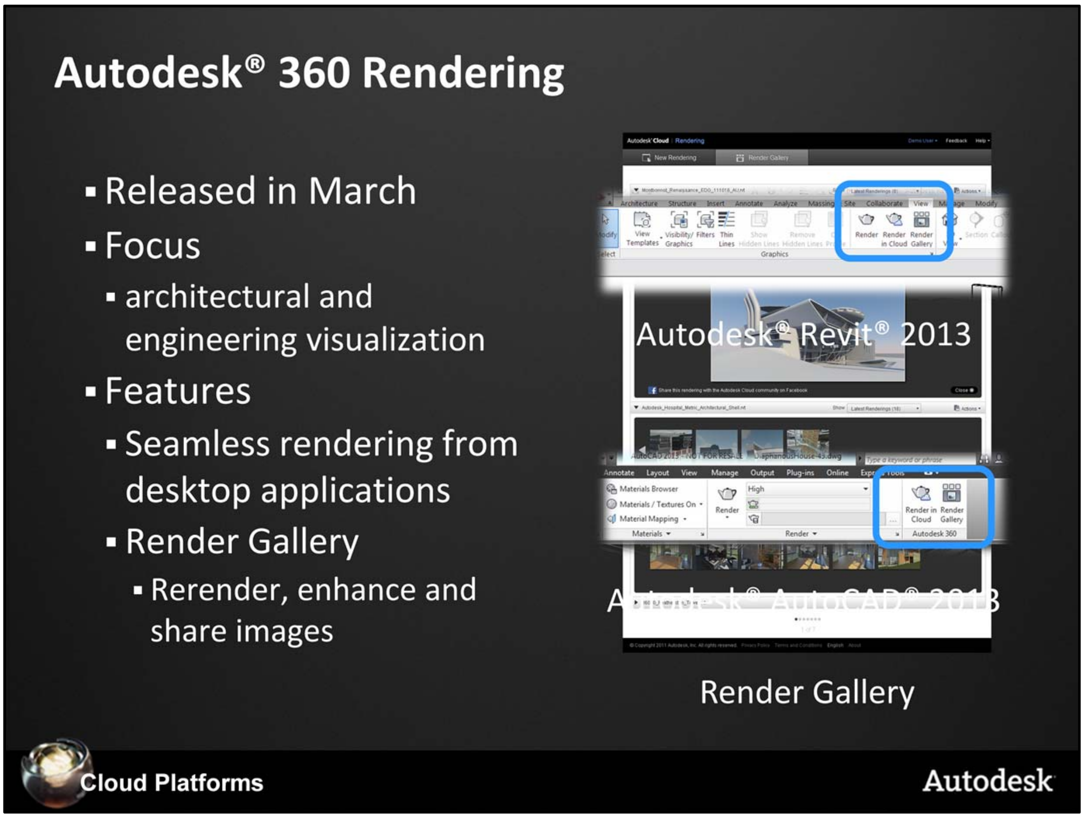
This final section of the course discusses the use of many lights algorithms in our cloud rendering service, Autodesk® 360 Rendering.

What is Autodesk 360 Rendering?

Cloud Platforms

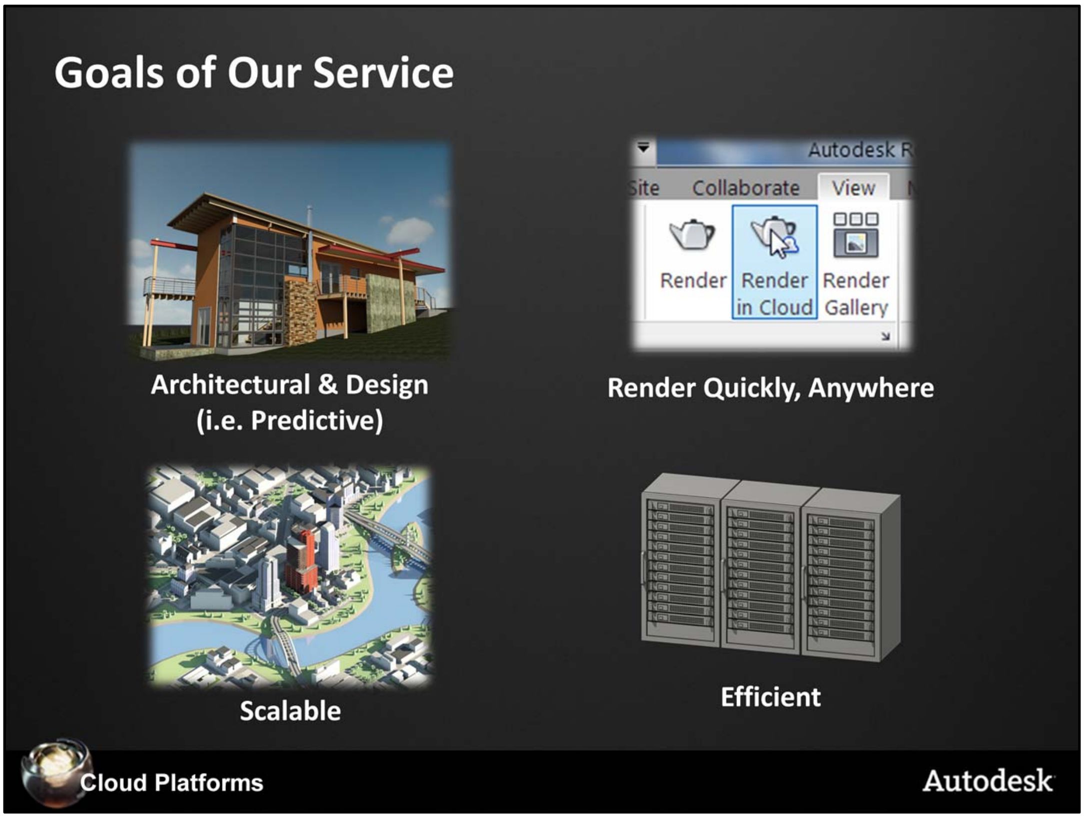
Autodesk

To begin I want to quickly introduce our service to illustrate the challenges that many lights algorithms help to us address. Our service is a component in a system of cloud applications, called Autodesk 360, launched last March. The goal of the system is to provide a suite of tools, accessible everywhere through web and mobile front ends, that allows users to store, share and collaborate on projects using our software. Our desktop applications are integrating with this cloud system to enable a seamless transition between local work and the cloud resources.

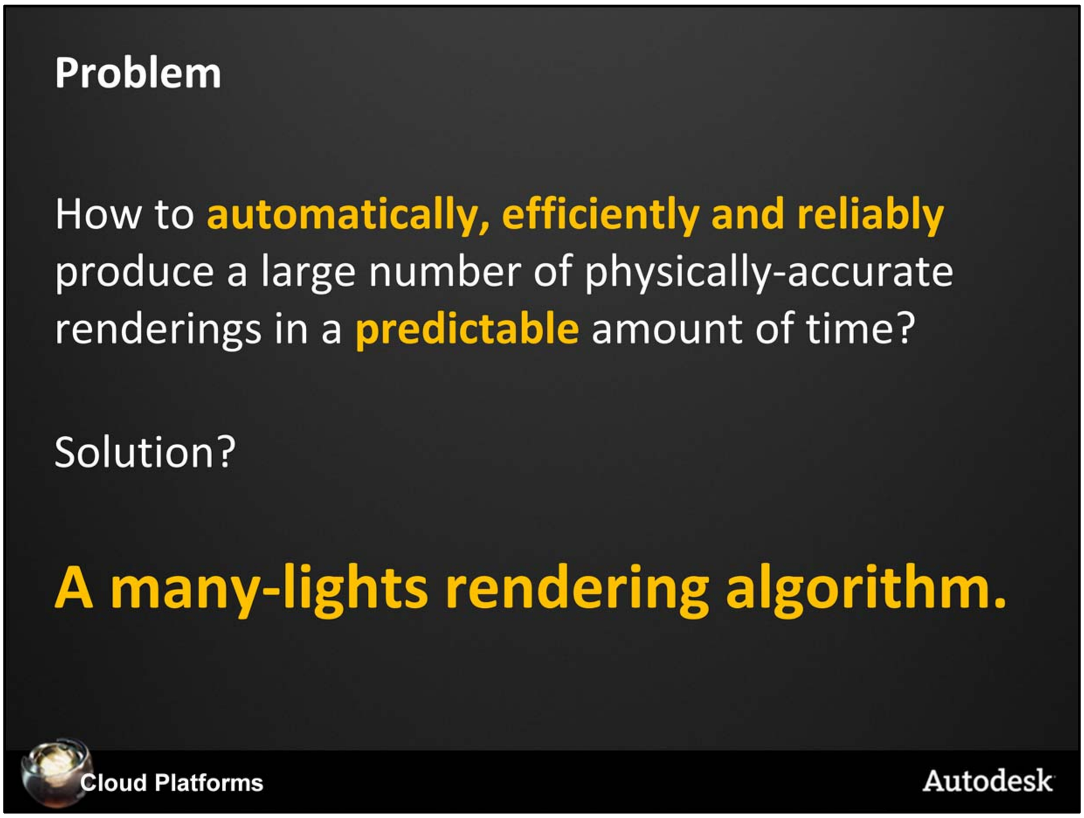


For rendering specifically, we have added cloud rendering options, mirroring to the previous desktop rendering options, to our Autodesk® Revit® and Autodesk AutoCAD® applications. Users can use these cloud rendering tools to upload and render their models remotely in the background. Completed renderings become available in our render gallery website where they can be viewed, modified and re-rendered with a list of advanced features, such as panorama views.

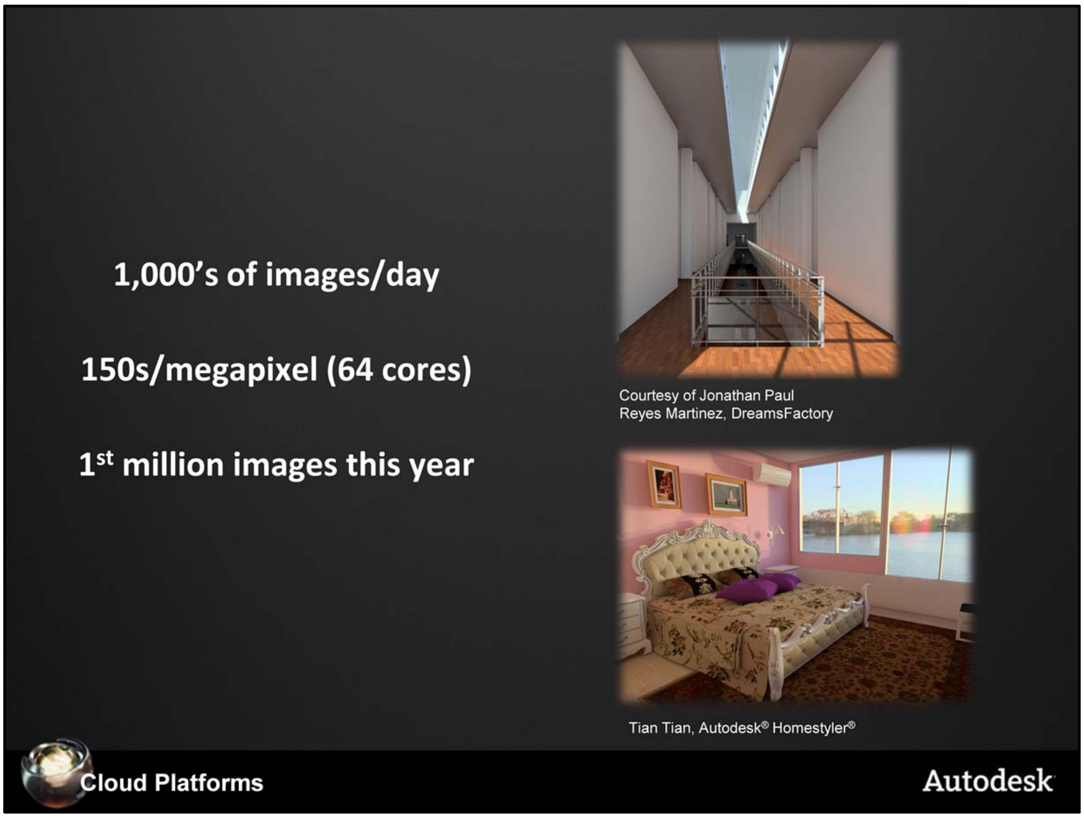
In addition our service provides back-end visualization support for consumer products such as Autodesk® Homestyler®, a tool lets users build and render their own interior design plans.



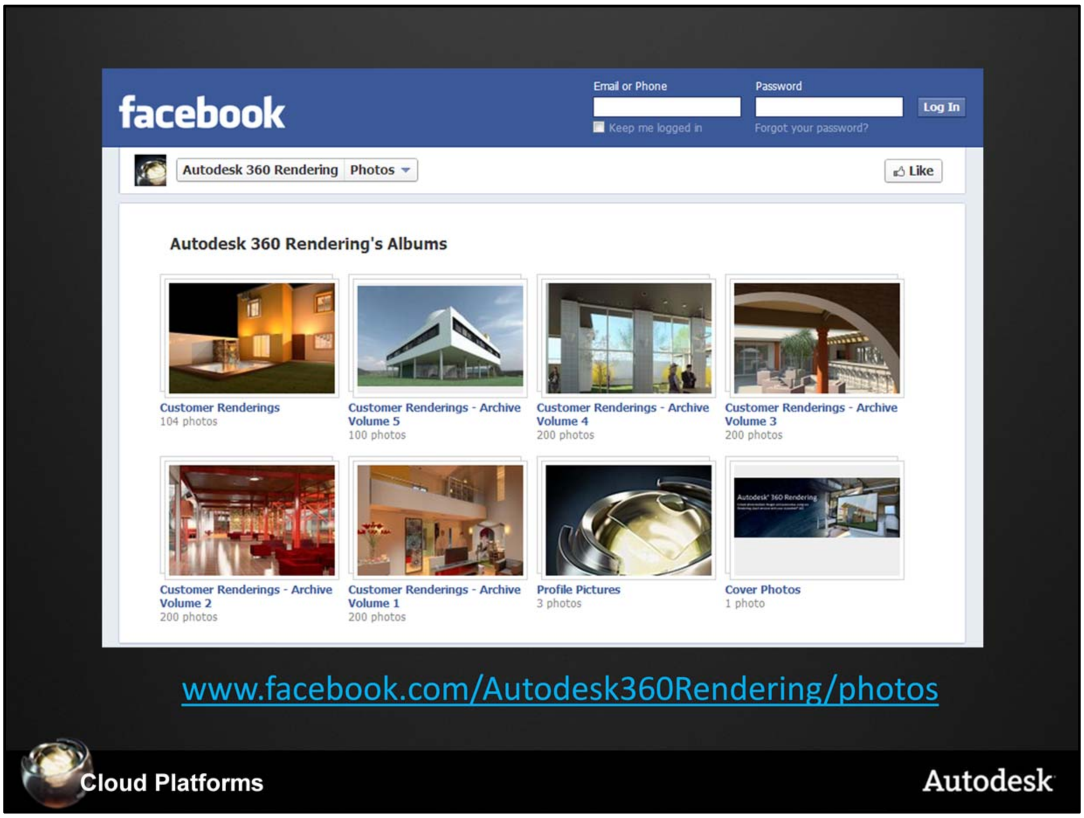
Working back from these applications, we can sketch out the goals of our cloud rendering system. First we focus primarily on architectural and design visualization. This focus is both challenging, because these applications demand higher-quality, predictive simulations, and simplifying, because these applications use a reduced palette of materials, lights and geometry that are generally physically modeled. Second, we want to make rendering a one-click option available anywhere in any product. This allows us to support consumer applications, such as Homestyler, and challenges us to make rendering simpler and easier to use. Third, we need our renderer to scale. Our users turn to the cloud to render their largest and most complex scenes, those beyond the capacity of their desktops, and they expect results quickly. And, finally of course, we need the renderer to be efficient since Autodesk bears the cost of the cloud compute resources.



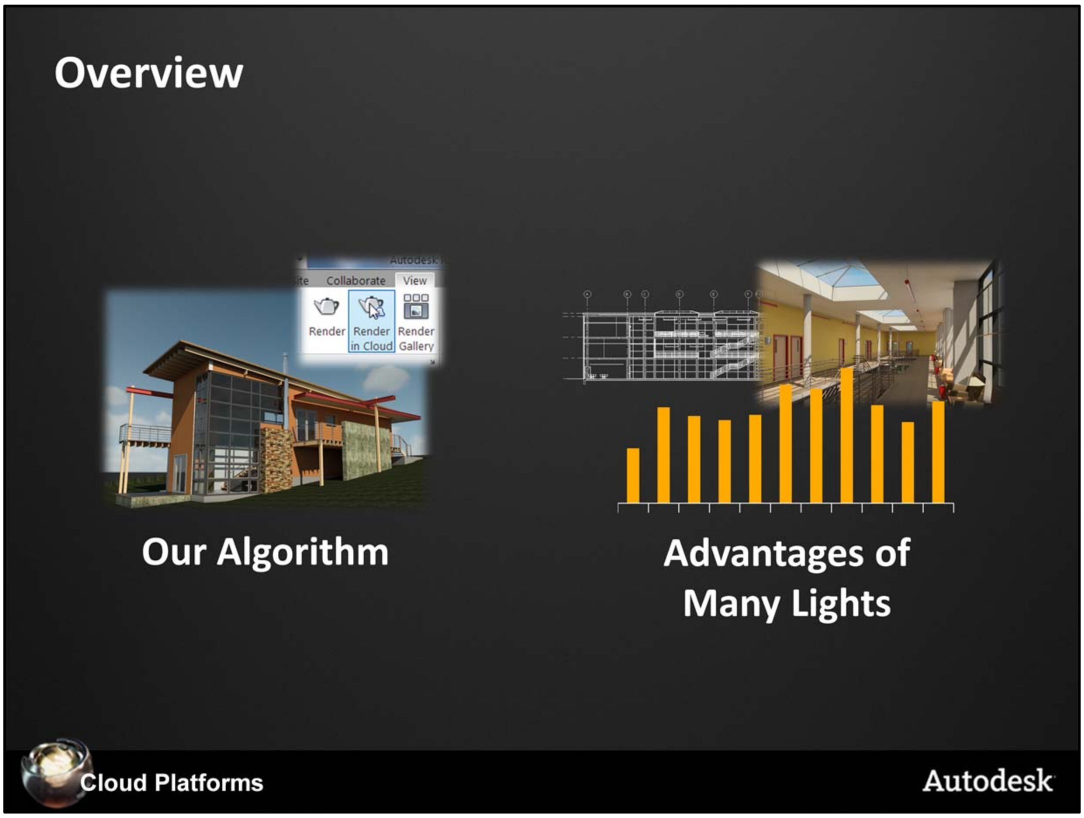
Now, meeting these goals can be consolidated into the central problem of our cloud rendering application: how can we automatically, efficiently and reliably produce a large number of physically-accurate renderings in a predictable amount of time? For our application, the solution to this problem was largely to use a many lights rendering algorithm. This talk discusses how we implemented many lights rendering in our system and why is was critical to our success.



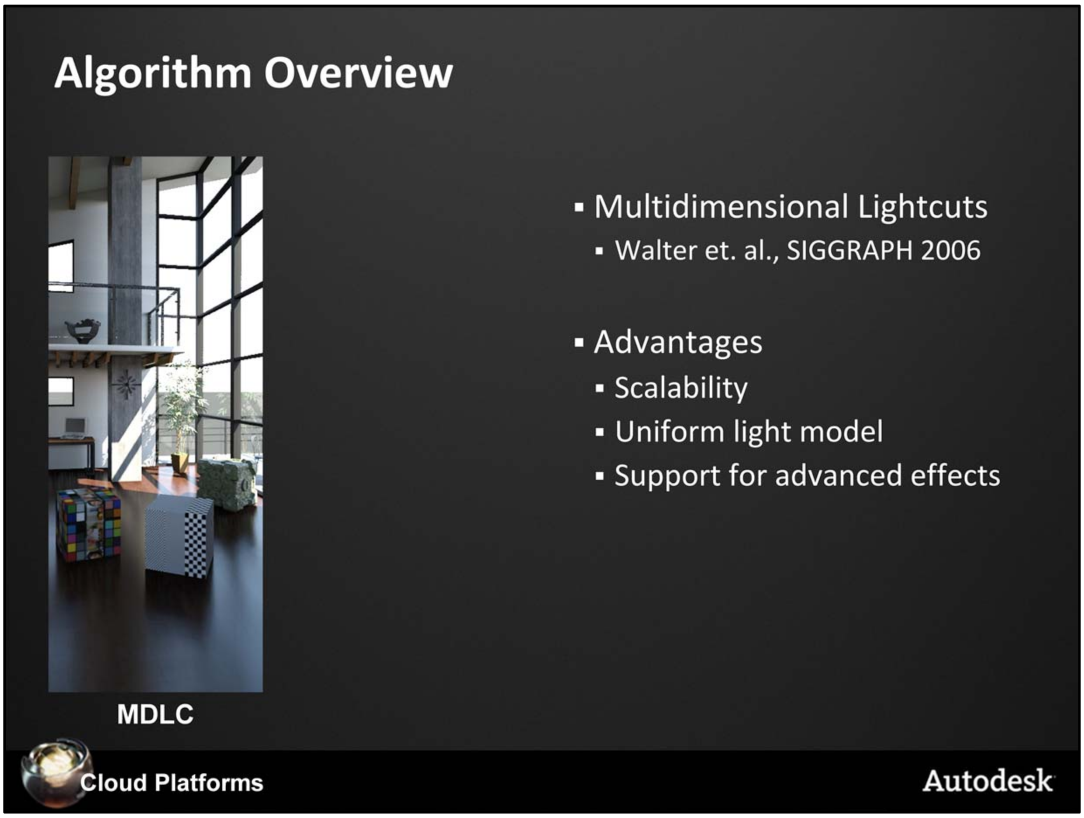
The results from our first few months have been robust and promising. We produce images using small clusters. Globally the cluster size is heterogeneous but on average we allocate 64 cores per image and produce a megapixel in 150s. Every day we render several thousand images and we should produce our millionth image sometime this year. The images on this slides are typical results shared by our customers.



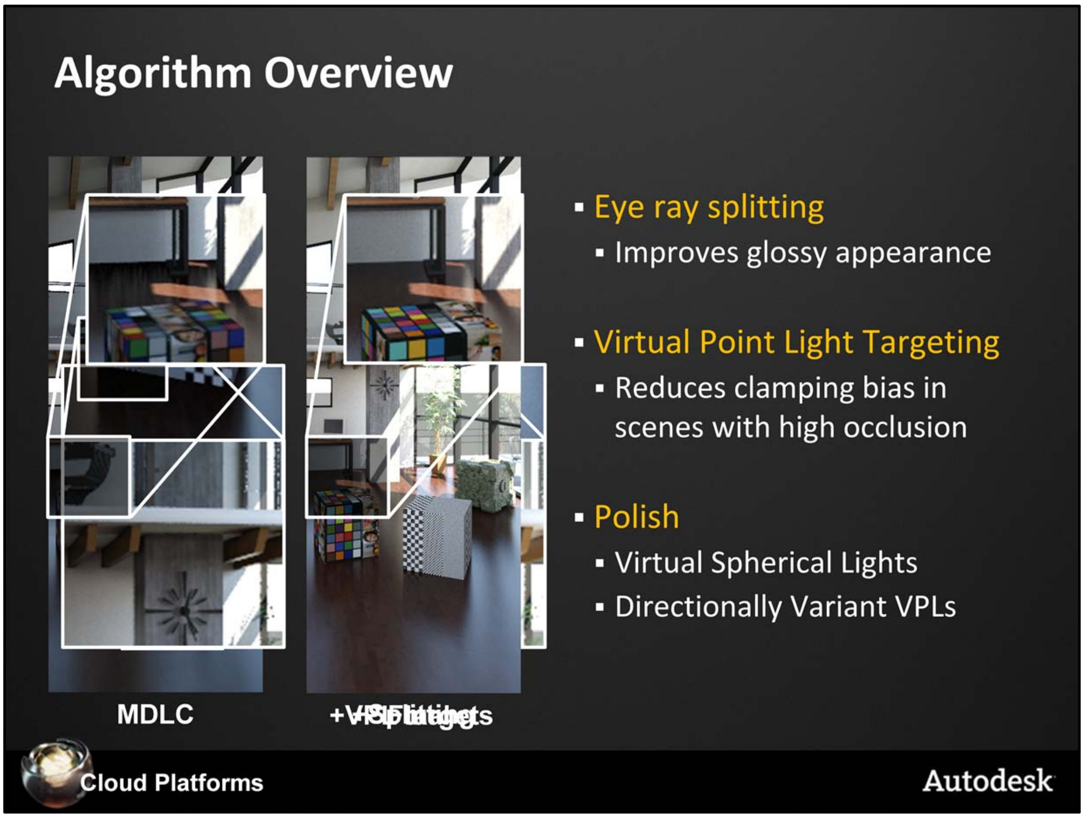
You can see images rendered with our service on our Facebook® page (as of mid-May 2012).



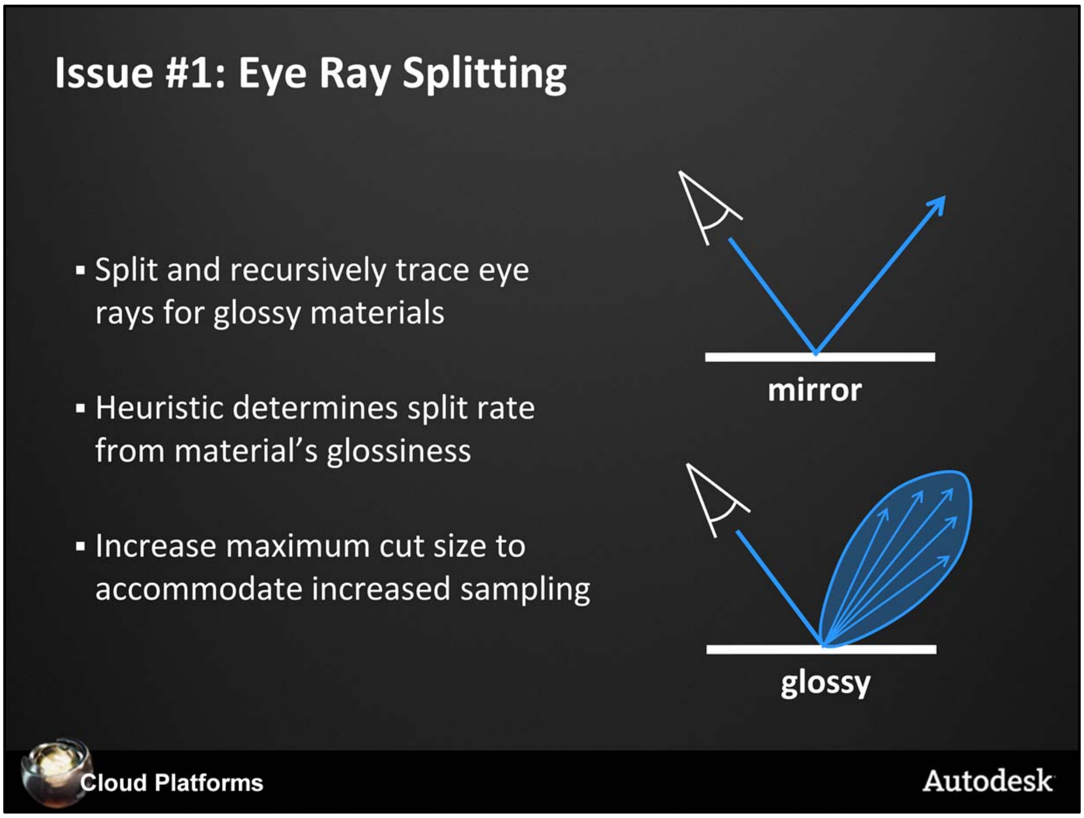
This talk will have two parts. The first part discusses the rendering algorithm we built at Autodesk and some of the implementation issues we addressed when developing that system. Then the second part discusses the advantages a many lights rendering algorithm brought to our application. The overall point of this second discussion is that many lights algorithms have proven to be faster and are fundamentally more scalable. Our results show that this holds true across a very wide array of images and scenes. However, since raw performance has been discussed at length in this course, this talk will focus on additional consequences of that scalability. Specifically, the goal of this talk is to describe how these algorithms also improve the reliability and predictability our rendering system and how their advantages have been critical to the success to our service. They have helped us to make rendering easier for novice users, to provide more consistent results for our customers and to improve the quality of images under fixed cost constraints.



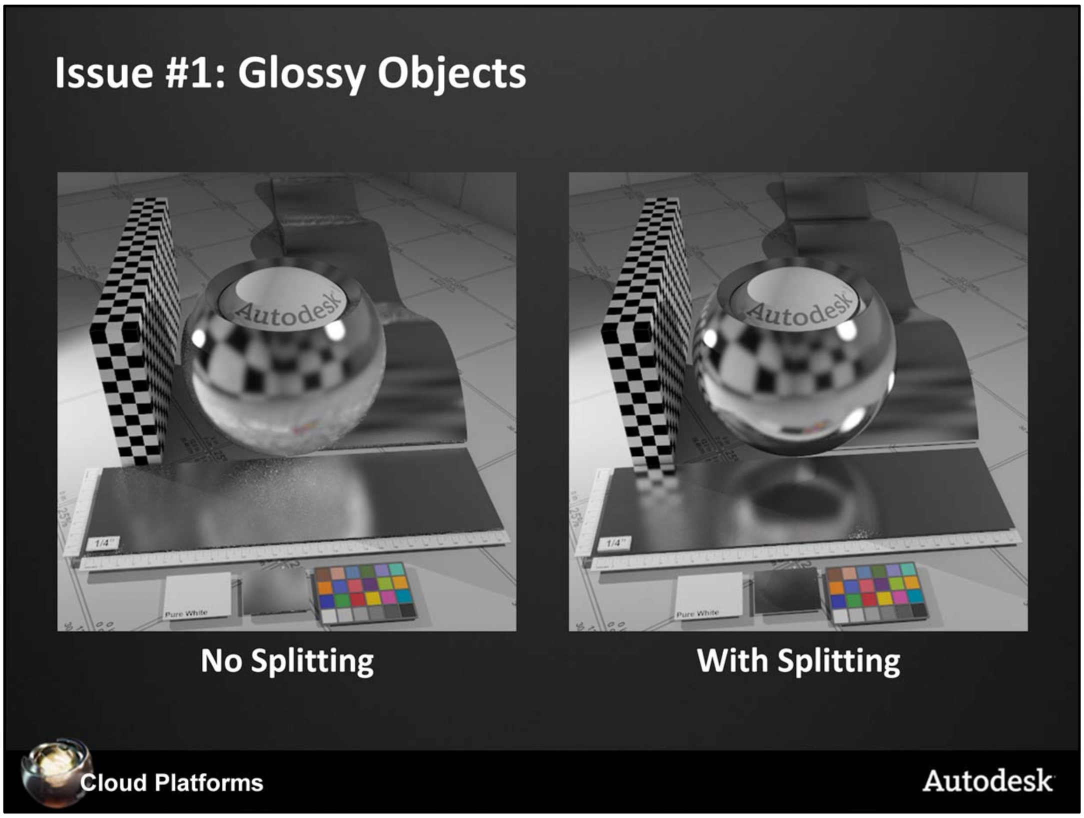
Our service uses Multidimensional Lightcuts to compute its images.



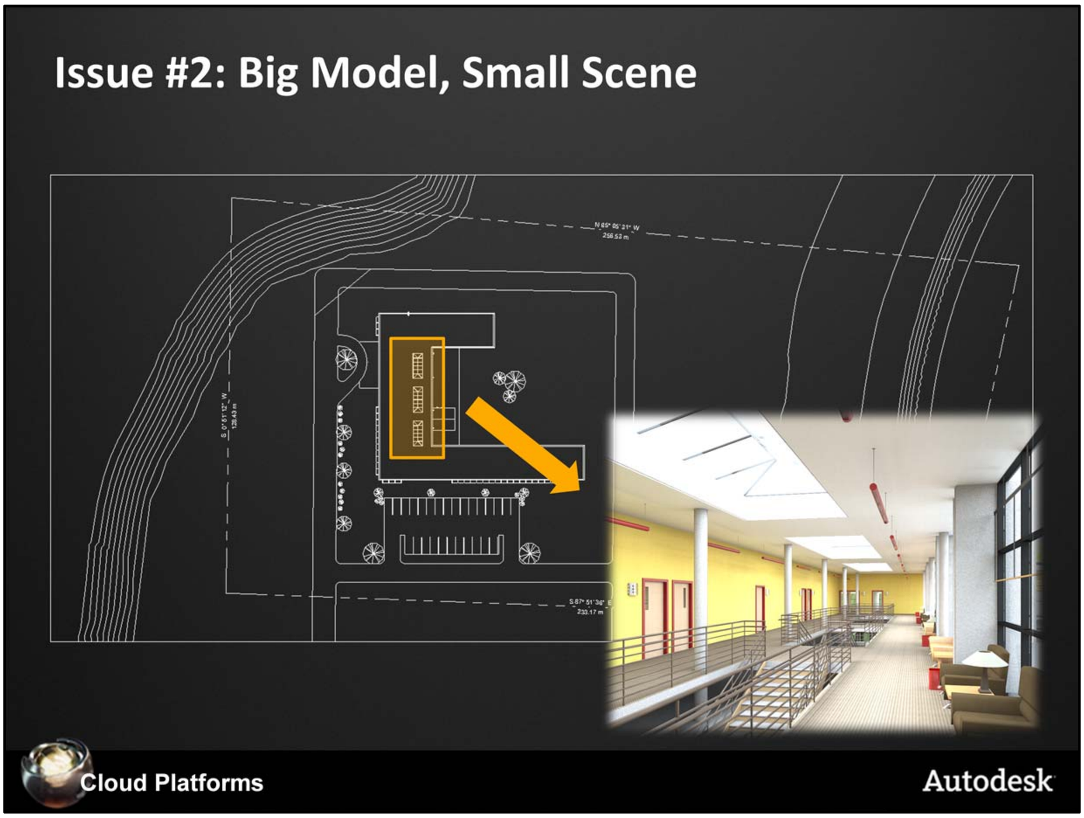
We implemented virtual spherical lights instead of VPLs to avoid some clamping and reduce the appearance of corner darkening. Our basic implementation is essentially identical to those described earlier in the course. However, in implementing these algorithms, we needed to address three important issues. First, we needed to add some form of bias compensation to better render glossy materials, particularly when there are glossy inter-reflections. We introduce an eye ray splitting heuristic solve this problem. Second, many scenes had difficult lighting occlusion and we faced issues generating robust VPL distributions. To generate more efficient VPLs sets, we added a VPL targeting method. Finally, we needed support for directional point light emission so we created a simple directional VPL type.

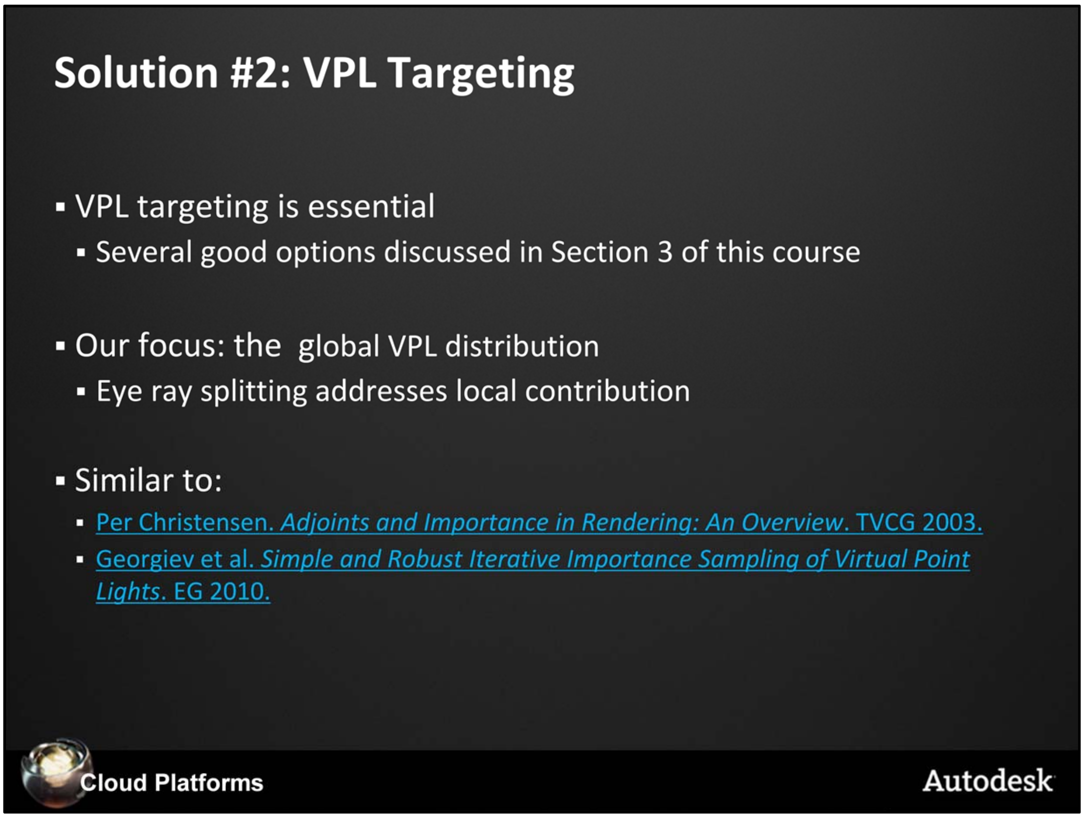


This recursion is identical to the recursion used for delta specular materials. When an eye path hits a sufficiently glossy surface, instead of creating a gather point immediately, we sample secondary rays and continue. The number of secondary rays and the decision to split is determined by a heuristic. Unfortunately, splitting does increase cost significantly; requiring an increase in the allowed maximum cut size.

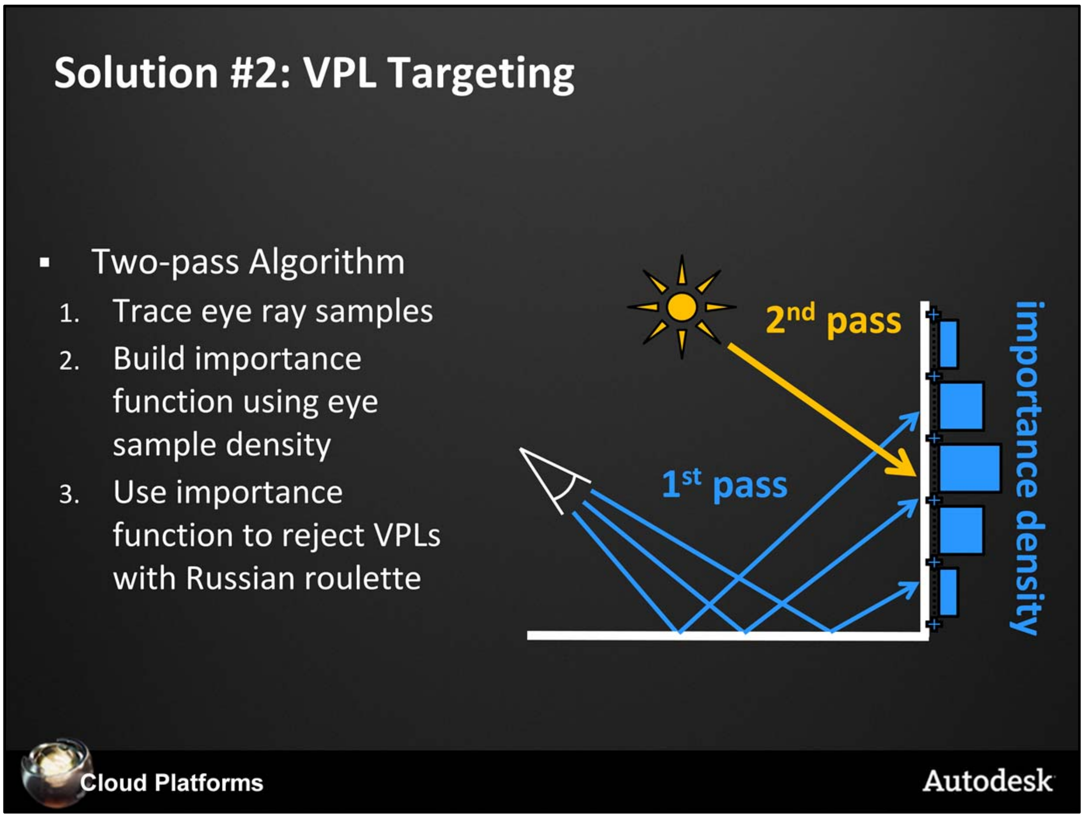


High gloss materials are common in our scenes. We found that clamping, sufficient to avoid noise, could significantly effect a material's appearance and that VSLs alone could not solve the problem (see the image on the left above and note the missing reflections of the glossy highlights in particular). To address this problem, we recursively continue our eye paths for glossy materials.

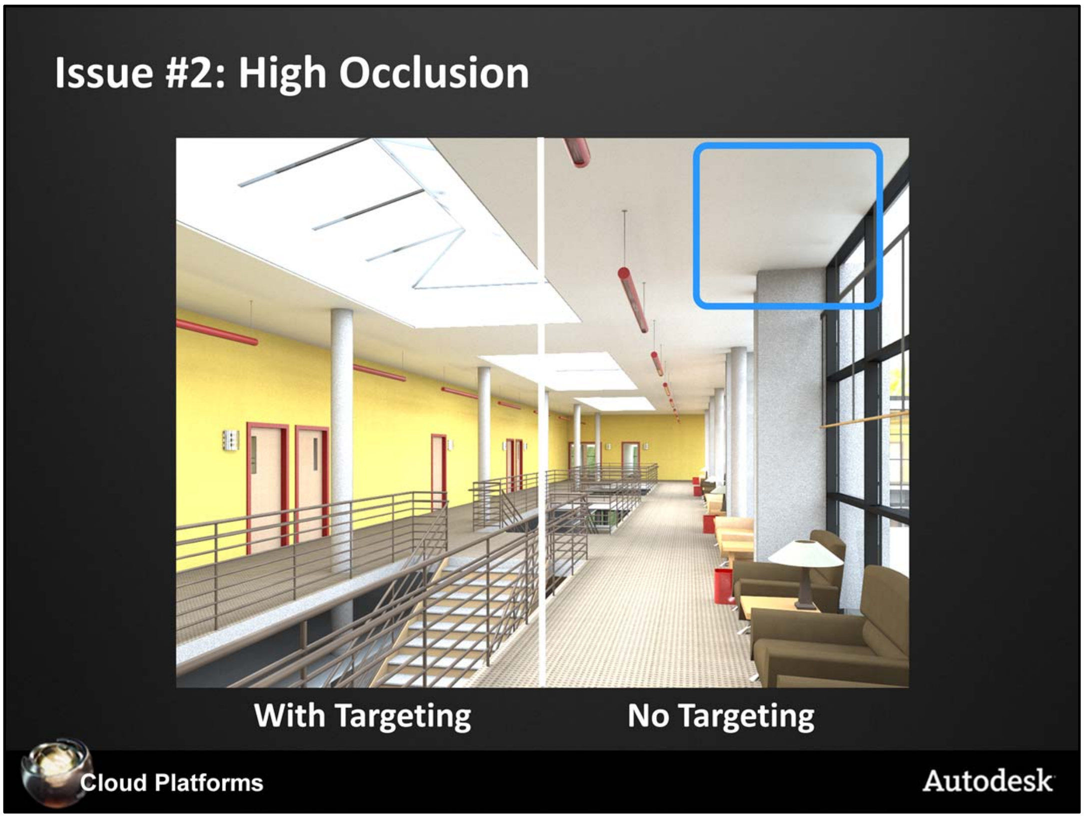

Issue #2: Big Model, Small Scene
Cloud Platforms
Autodesk



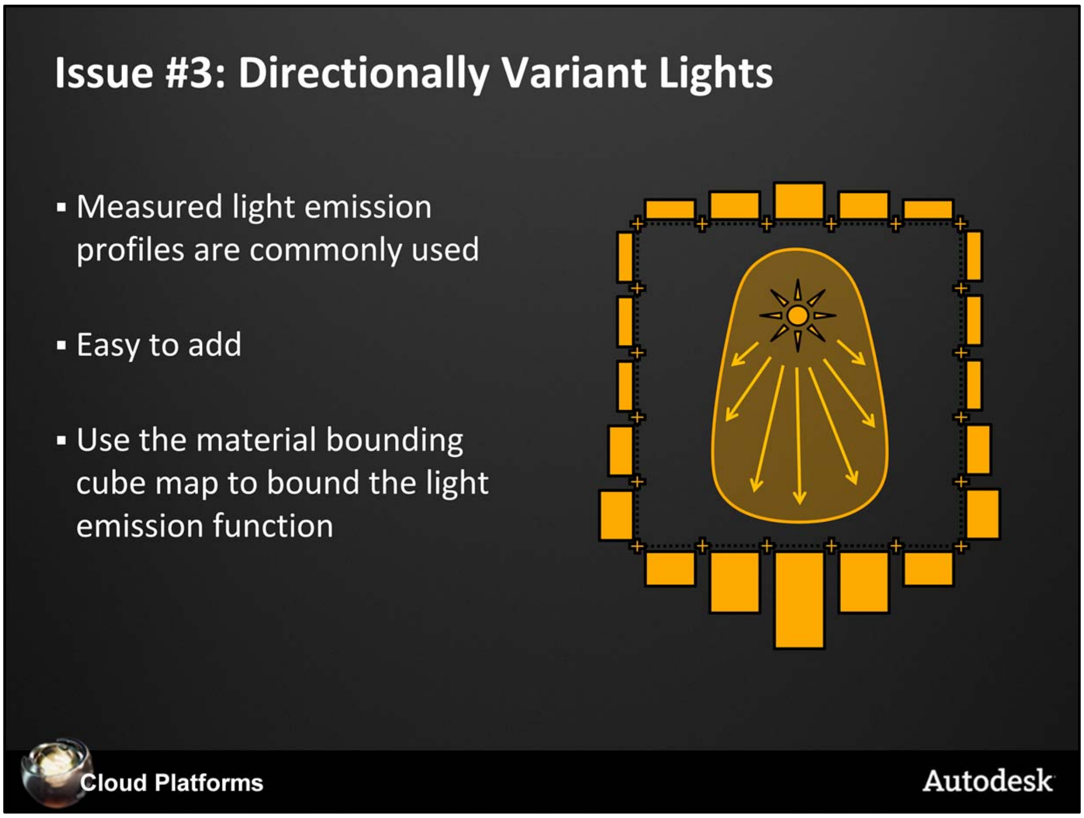
Our targeting method uses image importance to estimate the quality of a potential VPL. This method is modeled on methods for photon mapping with importance discussed by Per Chirstensen in TVCG 2003 and the method by Georgiev, et al. from EG 2010.



Our targeting algorithm splits the VPL tracing into two passes. In the first pass, importance carrying paths are generated from the eye. Exactly as in photon mapping, the resulting set of intersections ("importons") is stored and placed in a hierarchical acceleration structure. During the second VPL tracing pass, this importon map is queried to estimate the local importon density and we use Russian roulette to reject VPLs in low density regions.

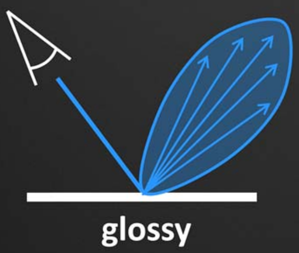


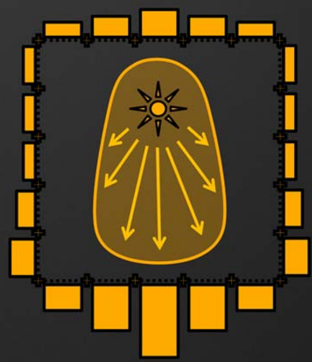
Our second issue was large scenes with significant occlusion to the lighting. In this example, we have a large office building model. In this image we are viewing only a single hallway in the model. Without modification to the VPL tracing algorithm, it is difficult to generate a sufficient density of VPLs in this region; instead VPLs are distributed over large regions of the model out of frame. As can be seen on the left side of this image, these low densities result in a negative, darker bias and banding artifacts on the ceiling. To address this issue, we resample the VPLs to ensure a higher density near the camera. To do this, we use a VPL targeting method to redistribute these VPLs. Empirically we have found that this targeting reduces bias, improves quality and reduces cost.



Finally, many of Autodesk's products support directionally emitting point lights and we had to implement a VPL type that could represent them. We found that the omni-directional VPL described in the original Multidimensional Lightcuts paper could be trivially extended to model directional emission. The only issue was bounding the light's emission function. To do this, we repurpose the cube maps that bound material reflectance also bound emittance.

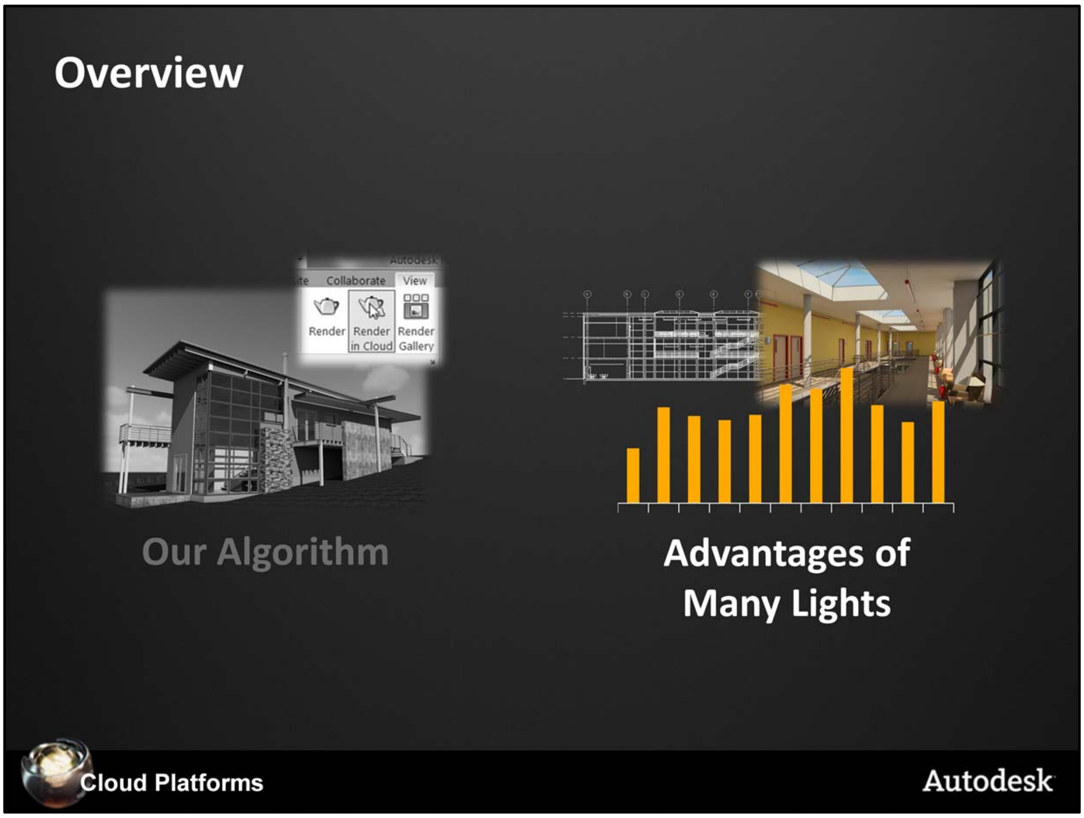
Formalized in Bidirectional Lightcuts





Bruce Walter, Pramook Khungurn and Kavita Bala, Cornell University
Light Rays
TUESDAY, 7 AUGUST 2:00 PM - 3:30 PM | Room 502AB

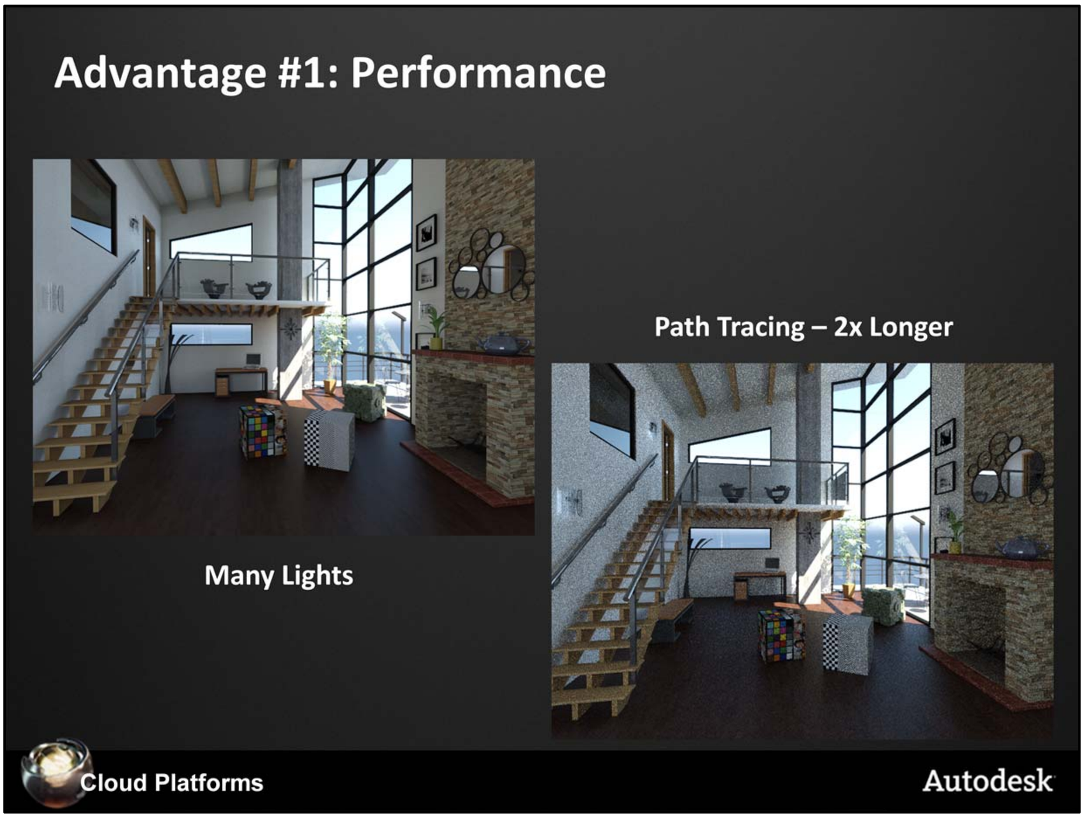
Cloud Platforms Autodesk



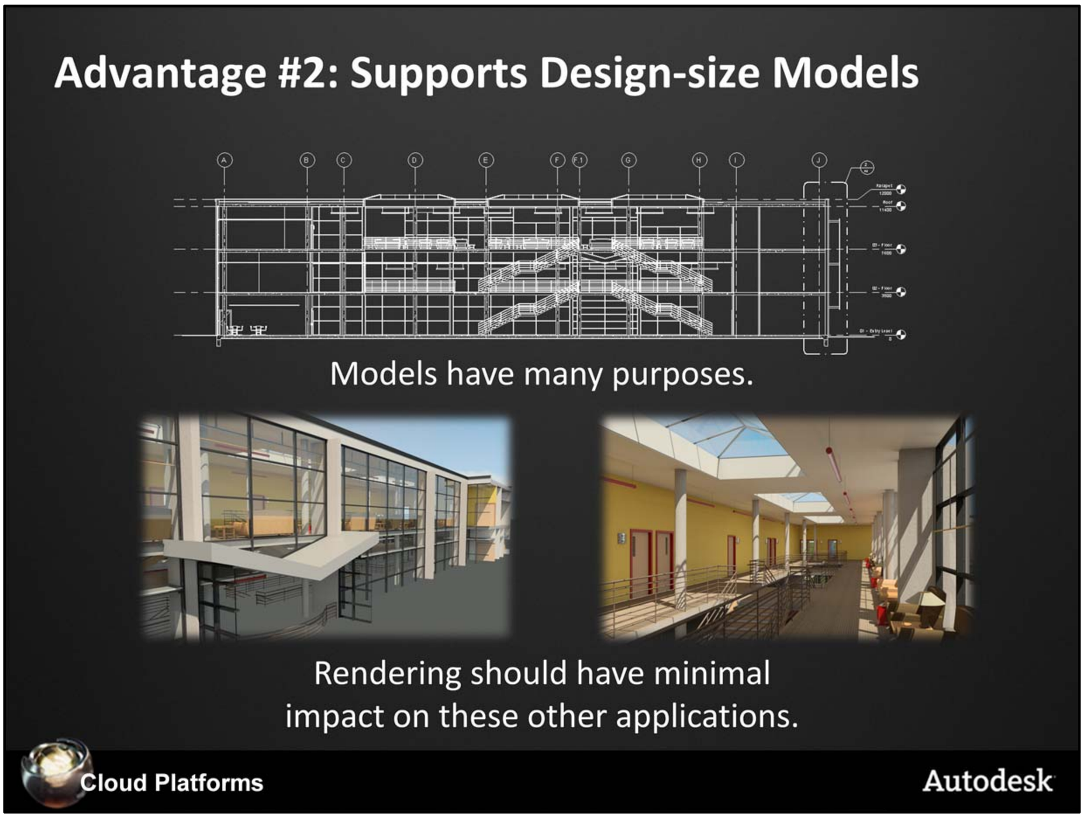
In the second part of this discussion, I want to highlight the advantages of the many lights rendering algorithm in our application.

Advantages of Many-Lights algorithms

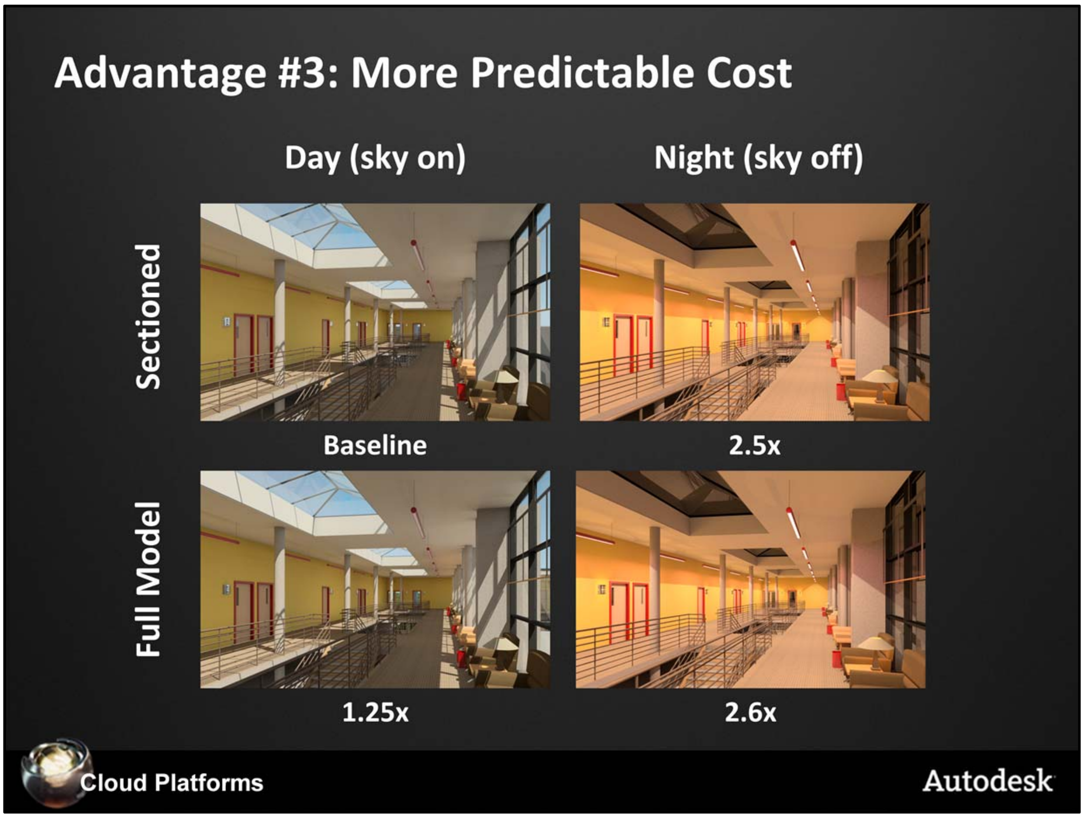
1. Performance
2. Robust to design size models
3. Automatic render setup
4. Predictable cost
5. High quality preview



Undoubtedly, the most important advantage of the many lights system is its performance. This point has been emphasized throughout this course but I want to reiterate it one last time. The performance advantages these algorithms have been significant and meaningful to our application for Autodesk. Not only has the performance impressed our customers but it has significantly reduced our costs. Every CPU/second our renderer saves is a CPU/second we don't have to buy and these algorithms save a lot of them. This is extremely valuable. However, since the performance has been stressed throughout the course, I want to take the remainder of this talk to discuss several additional advantages of these algorithms.

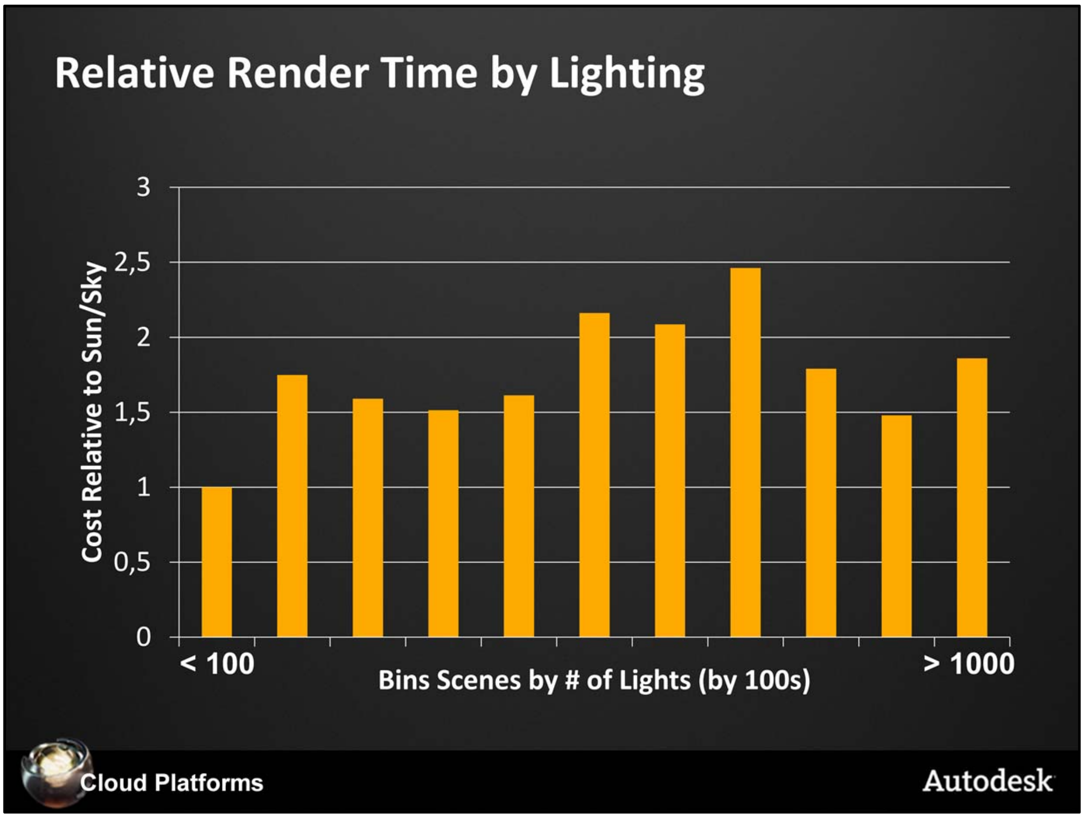


The third advantage of many lights technology is related to the last. Our users are building models for a diverse set of purposes. For example, architects and engineers use our software to make products or a buildings. Rendering these models should not interfere with these other purposes but this is not always true. Making a rendering can require the user to annotate daylight portals, create sectioned models (shown here bottom left) or enable/disable lights. However, the scalability of many lights algorithm bridges this render model/design model gap. The scalability lets our system robustly support “design” size models, reducing user effort and encouraging them to use rendering more.

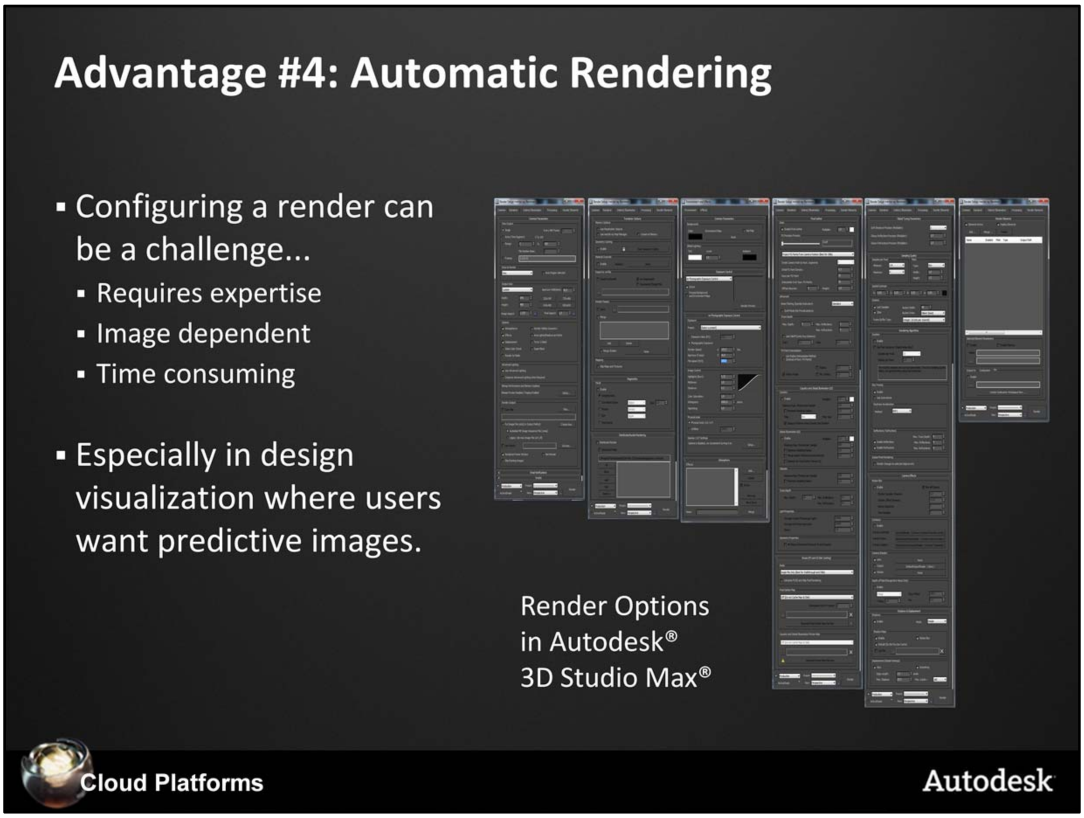


Additionally, the scalability of many lights algorithms makes rendering costs more uniform and predictable. This is important because our users want to use rendering to explore visual design options: lighting, window placements and building layout, for example. But this is harder to do if there is a huge variation in the cost of rendering these options. By making rendering more scalable and uniform, many lights algorithms allow our users to more freely render intermediate designs and enable them to use rendering to inform their designs rather than just to visualization the final choices.

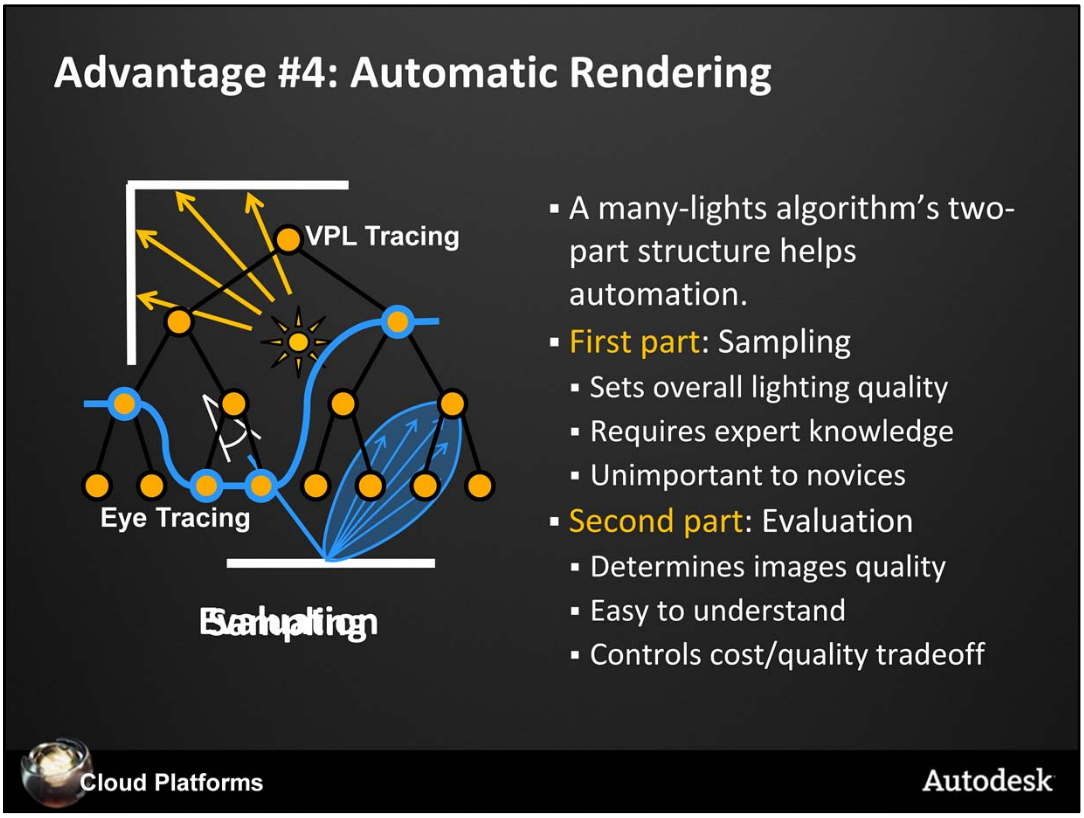
To illustrate this point, I rendered the same scene four times. In the left column, the procedural sun and sky is enabled and in the right it is off. In the top row, the model has been sectioned so that only the visible region is used for simulation while in the lower half the full model is preserved (including many light fixtures out of frame). Of course, costs increase with the increased geometry and lighting but they do so slowly. The increased render time is far outweighed by the time the user saves building the sectioned model or disabling light fixtures.



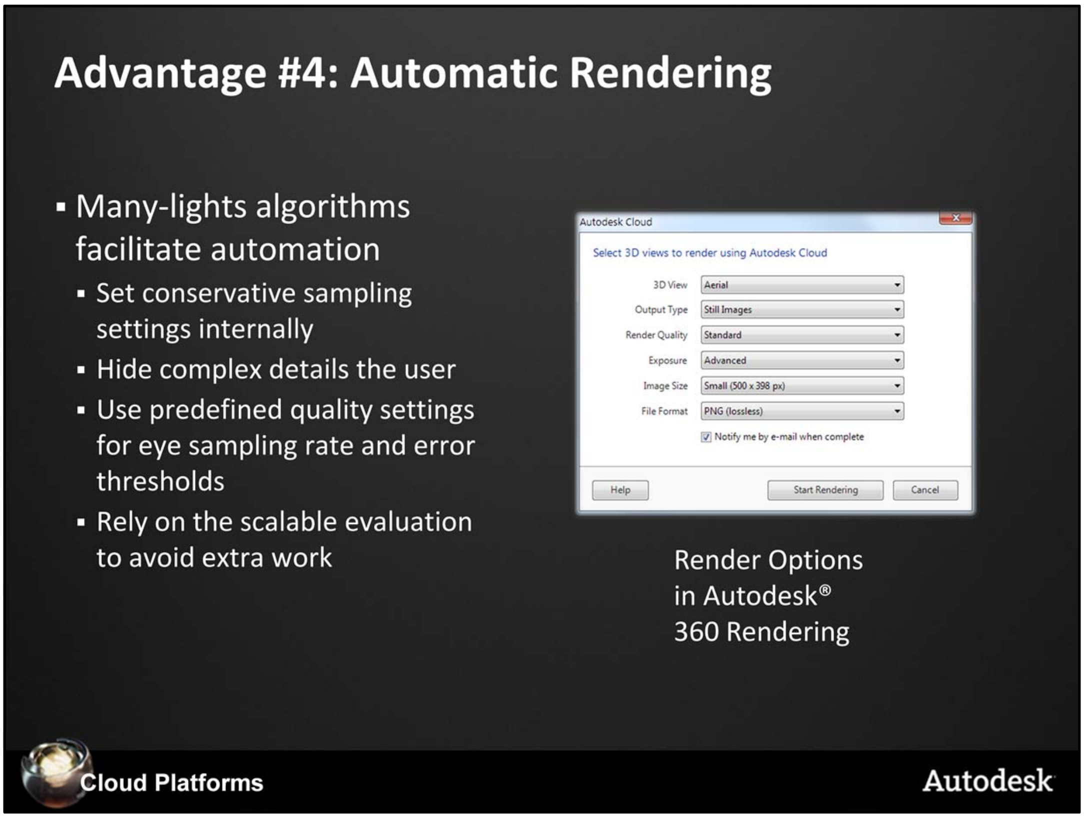
But one example is not as compelling as a thousand examples. I can continue this uniformity argument to a database of approximately 20K images. Here I plot the relative cost of adding lighting fixtures to scenes. In this sample, it turns out that sun/sky models have the lowest cost so I normalized this plot to show the relative cost of rendering models with only sunlight compared to those with both sunlight and fixtures. The models are grouped into bins of 100 fixtures each. The leftmost bin includes those scenes with less than 100 and the rightmost is scenes with more than 1000. Overall the results show remarkable consistency in the rendering cost across the whole range. *This* is the advantage of a many lights approach.



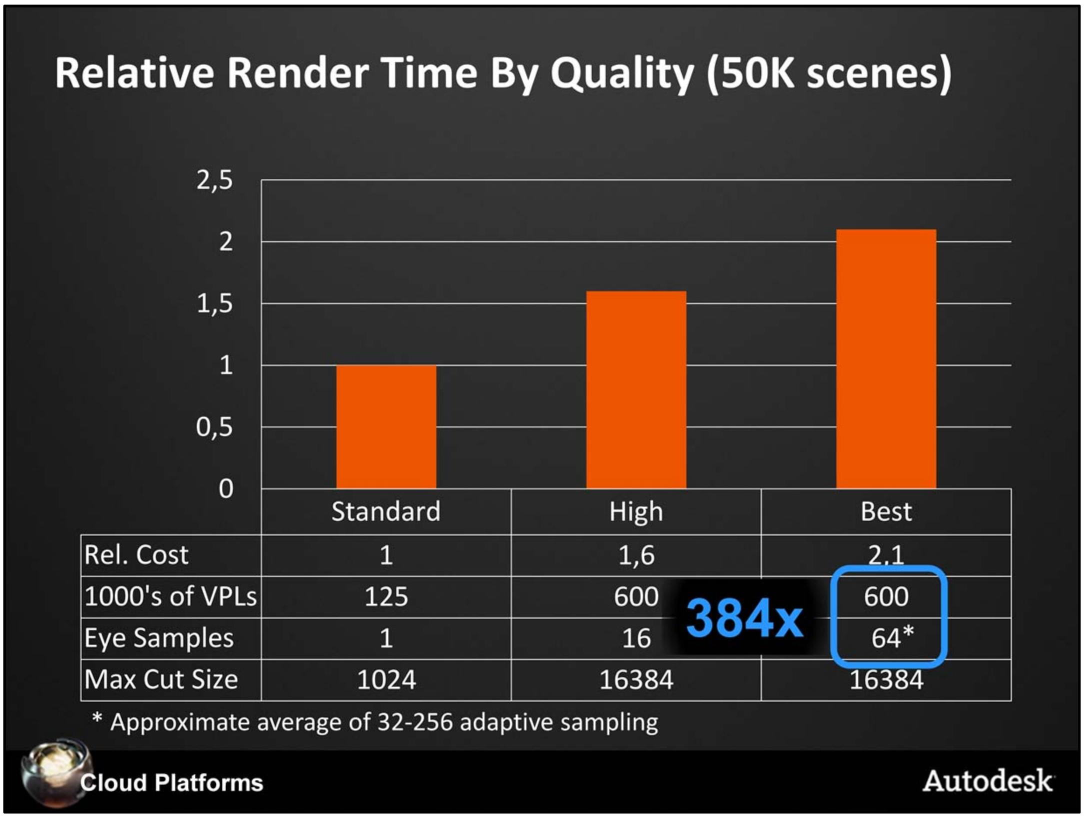
The first of these is that many lights algorithms are easy to automate. You can leverage their scalability, not just to make the rendering faster, but also to make the process of rendering easier. First, I note that configuring a render is a challenging task. On this slide, I have lined up the render options available in Autodesk® 3D Studio Max® (there are a lot of them). For an expert user, the complexity of these controls is extremely useful. They can tune the renderer to achieve an certain artistic look, tailor the simulation to be maximally efficient for a particular scene and selectively downgrade the computation for pre-visualization. However, for novice users these controls are confusing and, moreover in our cloud application, they are somewhat unnecessary. Users assume that in the cloud there are enough resources to compute their image and they are less concerned with performance tuning options. Our users just want a certain predictive quality without a lot of effort.



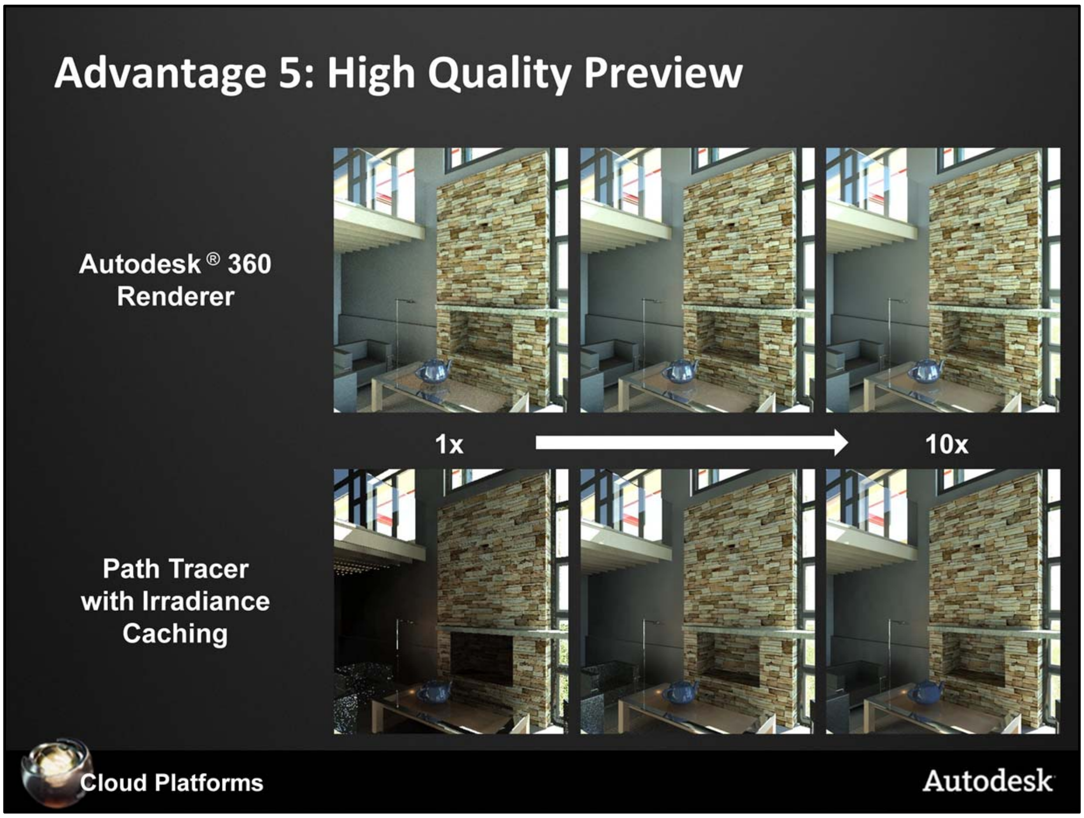
Conveniently the structure and scalability of many lights algorithms make this easy to achieve. One can divide a many-lights algorithm into two components. The VPL tracing and the eye ray generation represent a sampling component. The evaluation of the samples to generate an image is a second component. Largely users care about the how the second component behaves. It encapsulates an intuitive quality/cost tradeoff. However, only expert users understand how to correctly tune the first sampling component. Most users just want it to be set "correctly".



With many lights, we can do exactly this. In our application, we can permanently set conservative, "correct" VPL and eye sampling parameters internally in the renderer. This hides detailed mathematical information about our algorithm from the user. Instead we let the user control quality and time by choosing from a predefined set of quality choices controlling the error-bounds, cut size and sampling rates. This works because we can rely on the fundamental scalability of the renderer to avoid extra work if our sampling is a little too conservative. This lets the user focus on more intuitive parts of their request (see our render settings dialog) such as image size, quality and format.

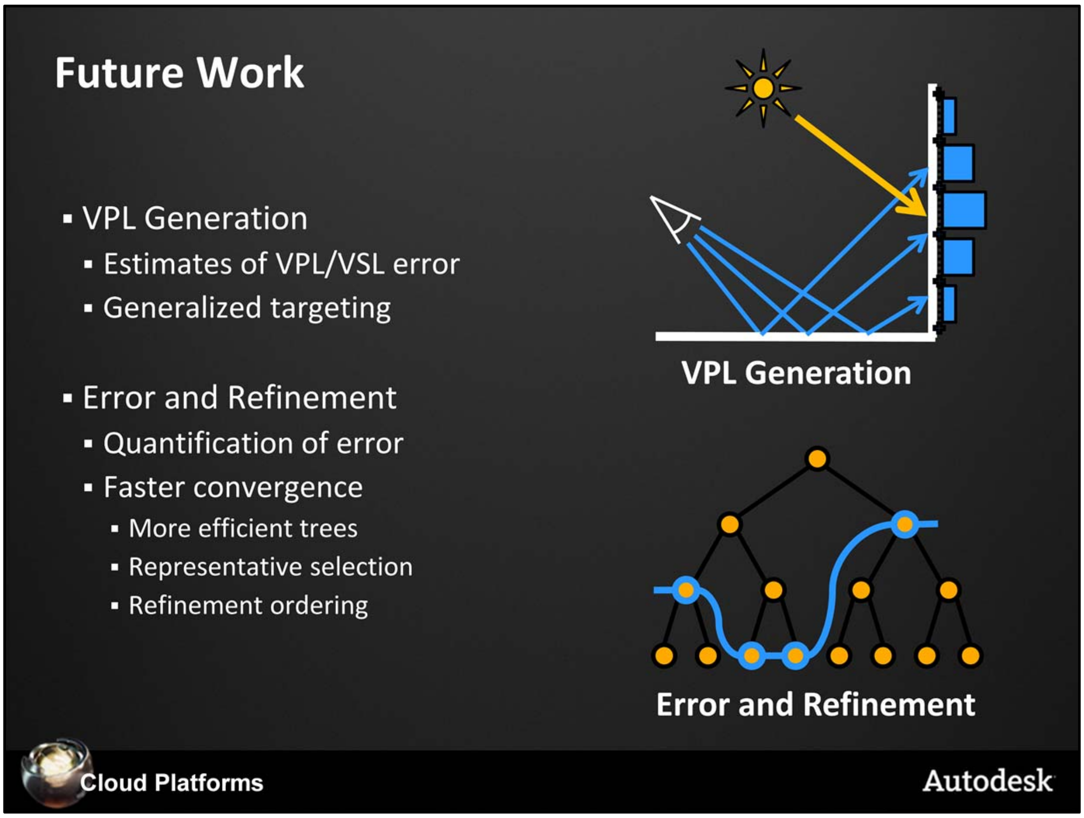


I can demonstrate that this works well in practice. Here we look at the relative render times per megapixel for approximately 50,000 jobs rendered in April 2012. The results demonstrate two important features of a many lights algorithm. One, fixed conservative setting reliably produce images across a wide range of scenes. Two, the algorithms robustly avoid extra work: the highest quality is, on average, only 2x the cost of the baseline despite many more VPLs, eye samples and a much larger maximum cut size.

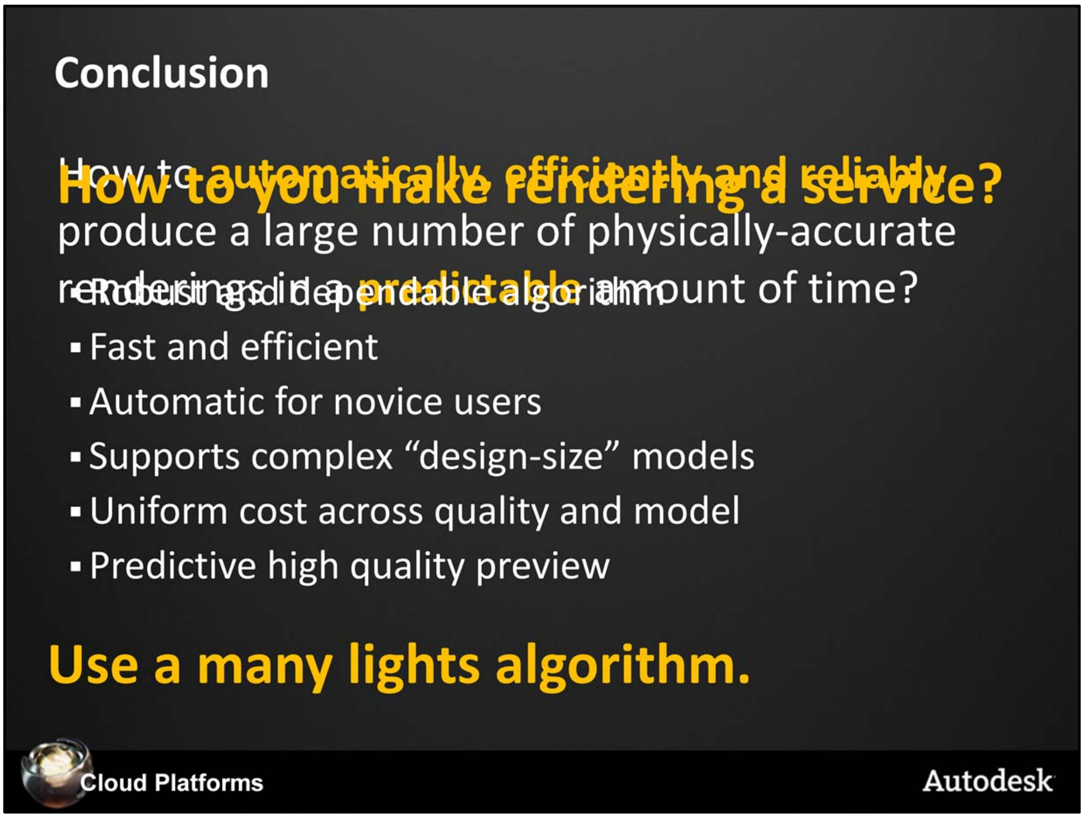


Finally, the last advantage is one of the biggest: the performance of the algorithm at low quality. This is very critical for our cloud application because we need to offer cheap, fast renderings, useful for intermediate feedback, that are predictive of a long-running, high-quality final result.

This slide compares this cost/quality tradeoff for our system (top) and a path tracer with irradiance caching (bottom). The images on the right are final quality and cost 10x more than the images in the leftmost column. (Note: columns are equal time comparisons between the two renderers; and the underlying material system for the two renderers is not the same so the teapot and table appear slightly different.) Because we can keep the VPL sampling rate the same across this whole set (VPL cost is included in the timings), the many lights solution tends to preserve the lighting quality and appearance across the whole range. However, the path traced solution becomes negatively biased as the irradiance cache sampling rate and density become low.



Of course, the work is never done. To close I want to highlight two issues that would considerably improve our system. Foremost, general VPL targeting is an unsolved problem. Fundamentally, we need methods to assess the error of a particular set of VPLs and to generalize the targeting process to reduce this error. Second, there seems to be room for further performance improvements in the cut refinement process. One, if we could estimate the absolute error of an intermediate cut, we could begin to quantify the absolute accuracy of many lights methods. Also we want to continue to investigate whether efficiency could be improved by altering the VPL trees, representative selection or cut refinement ordering.



To conclude this talk, I return to the problem I discussed at the beginning. How to you make rendering accurate, fast, automatic, efficient and reliable? Or more succinctly: how do you make rendering a service? That is our goal at Autodesk.

Acknowledgements

- My Team
  - John Hutchinson
  - Roberto Ziche
  - Seema Jaisinghani
  - Brian Budge
  - Nolan Goodnight
  - William Xiao (肖健）
  - Jacky Lee (李吉超)
  - Yonggao Pan (潘永高)
  - Anders Huang (黄运新)
  - Mintao Huang（黄敏涛）
  - Xiaoqing Zhuang (庄晓青)
  - Ada Yan (严超)
  - Sally Dong (董月娟)
  - Winnie Yu (喻超华)
  - Rick Wu (吴懿)
  - Jieqi Ding (丁洁琦)
  - Stella Zhou (周翊)
- Presenters
  - Jaroslav Křivánek
  - Miloš Hašan
  - Carsten Dachsbacher
  - Alexander Keller
  - Bruce Walter
- Cornell University

Questions?

Cloud Platforms Autodesk

Autodesk®

Cloud Platforms

Autodesk